ONR Grant No. N00014-91-J-1011
R&T Project: 4148501---02

Semi-Annual Report

(2)

DEVELOPMENT OF PARALLEL ARCHITECTURES FOR SENSOR ARRAY PROCESSING ALGORITHMS

Submitted to:

Department of the Navy
Office of the Chief of the Naval Research
Arlington, VA 22217-5000

Submitted by:

M. M. Jamali Principal Investigator

S. C. Kwatra Co-Investigator

Department of Electrical Engineering
College of Engineering
The University of Toledo
Toledo, Ohio 43606

Report No. DSPH-2

July 1992

ABSTRACT

The high resolution direction-of-arrival (DOA) estimation has been an important area of research for number of years. Many researchers have developed number of algorithms in this area. Fast advancement in the areas of Very Large Scale Integration (VLSI), Computer Aided Design (CAD) and Application Specific Integrated Circuit (ASIC) design, has made possible the development of dedicated hardware for sensor array processing algorithms. In this research we have first focussed our research for the development of parallel architecture for Multiple Signal Classification (MUSIC) and Estimation of Signal Parameter via Rotational Invariance Technique (ESPRIT) algorithms for the narrow band sources. The second part of this research is to perform DOA estimation for the wideband sources using two algorithms. All these algorithms have been substituted with computationally efficient modules and converted them to pipelined and parallel algorithms. Parallel Architectures for the computation of these algorithms and architectures has been developed. Simulations of these algorithms and architectures has been performed and more detailed simulations are in progress.

Chapter 1 presents theoretical and mathematical aspect of MUSIC/ ESPRIT algorithms. These algorithms are modified and parallelized for narrow band case in chapter 2. Hardware implementations of these algorithms are in Chapter 3. Development of Generalized Processor - GP is shown in Chapter 4. In Chapter 5 DOA estimation for Broad-Band sources using "Broad-Band Signal Subspace Spatial-Spectral" (BASS-ALE) Estimation algorithm and its architecture is described. Chapter 6 gives the details of hardware development for the bilinear transformation approach for wide band sources. Data generation and simulation of DOA estimation both for narrow band and wideband cases are given in Chapter 7. Conclusions and future directions are described in Chapter 8.

Table of Contents

LIST OF FIGURES

Chapter 1

INTRODUCTION TO ARRAY SIGNAL PROCESSING

1.1 INTRODUCTION

The high resolution direction-of-arrival (DOA) estimation is important in many sensor systems. It is based on the processing of the received signal and extracting the desired parameters of the DOA of plane waves. Many approaches have been used for the purpose of implementing the function required for the DOA estimation including the so called maximum likelihood (ML) and the maximum entropy (ME) methods [1-3]. Although they are widely used, they have met with only moderate success. The ML method yields to a set of highly non linear equations, while the ME introduces bias and sensibility parameters estimates due to use of an incorrect mode (e.g. AR rather than ARMA). The Multiple Signal Classification (MUSIC) and the Estimation of Signal Parameters by Rotational Invariance techniques (ESPRIT) algorithms are two novel approaches used recently to provide asymptotically unbiased and efficient estimates of the DOA [4,5]. They are believed to be the most promising and leading candidates for further study and hardware implementation for real time applications. They estimate the so called signal subspace from the array measurements. The parameters of interest (i.e. determining of the DOA) are then estimated from the intersection between the array manifold and the estimated subspace.

An important aspect of the design of a signal processing system for the DOA is the computation of the spectral decomposition. In recent years, the search for useful algorithms and their associated architecture using special purpose processors

has been a challenging task. Such high performance processors are often required to be used in real time application; thus, it is felt that they should rely on efficient implementation of the algorithms by exploiting pipelining and parallel processing to achieve a high throughput rate. The QR algorithm is one of the most promising for the spectral decomposition problem due to its stability, convergence rate properties, and suitability for VLSI implementation [6].

A number of investigations have been concerned with finding efficient algorithms to solve the spectral decomposition problem based on the QR algorithm. These investigations have mostly relied on systolic arrays approach. A primary reason for employing such approach is that it is believed to offer a well-motivated methodology for handling the high computation rate required for a real time application.

A useful property of the QR transformations is that shifts can be used to increase the rate of convergence to locate the eigenvalues [7]. This may be very useful for some systems applications where the computations of the eigenvalues are sufficient, such as matrix rank determination and system identification. However, in other applications, (e.g. , direction of arrival estimation, spectral estimation, and antenna beamformation), the computation of both the eigenvectors and eigenvalues is crucial [4-8], and one might use the QR algorithm without shifts to obtain these parameters in parallel. In such a case, this algorithm may require a sufficiently large number of iterations to converge. Keeping the number of iterations low may yield to inferior results such as in MUSIC and ESPRIT algorithms, where an accurate computation of the eigenvalues and eigenvectors will also determine the accuracy of the direction of arrival (DOA's) . For example,

for the MUSIC algorithm [8], once we determine the signal and noise subspaces from the eignenvectors, the spacial spectra is determined by

$$S(\Theta) = \frac{1}{a^{H}(\Theta)\mathbf{E}_N \mathbf{E}_N^{H} a(\Theta)} \qquad (1.1)$$

where $\mathbf{E}_N$ is the matrix of eigenvectors spanning the noise subspace, and

$$a(\Theta) = [\, 1, \exp(-j\Theta), \exp(-2j\Theta), \ldots, \exp(-j\Theta(m-1))] \qquad (1.2)$$

with m being the number of sensors, and H denoting complex conjugate transpose.

Also , one drawback of the QR algorithm is that when applied to a dense matrix, it may be very time-consuming and may pose difficulties for parallel implementation due to communication and timing among different modules of the systolic array [9]. For this reason, currently, there is no known simple efficient systolic array approach using the QR algorithm that is capable of generating the eigenvectors and eingenvalues in parallel.

1.2 DATA MODEL

For the purpose of understanding the advantages of using a sensor array in DOA estimation, it is necessary to explore the nature of signals and noise the array is desired to receive. It is well known that in active sensing situations, the scattered data fluctuates randomly about a true value representing a noise free signal. This is due to noise effects and errors in a sensor array system. These fluctuations can be both additive and multiplicative. The additive fluctuations are due to thermal noise, shot noise, atmospheric noise, and other kinds of noise which are independent of the desired signal. The multiplicative fluctuations are due to measured errors in estimating the signal amplitudes, gain variation, etc. A noise

model that represents all these noise effects is, in general, difficult to obtain, especially when some of the noise sources are dominant. Usually, based on the noise models, additive and/or multiplicative, the calculated probability of error, as a function of the noise power, is practically similar in each case. This indicates that the noise power, rather than its specific characteristics, has more impact on the sensor array performance. Moreover, one is usually concerned with the effects of the additive noise on the output of a sensor array system. For this reason, an additive noise would be appropriate to choose for the evaluation of the performance of a system. This noise represents the totality of small independent sources, and by virtue of the central limit theorem one can model the resulting noise as Gaussian and (usually) stationary process. Also, to make the problem analytically tractable, first narrow band signals are considered where it is assumed that the power of all emitter signals is concentrated in the same narrow frequency band. In this context, two more assumptions that are of interest are invoked. First, it is assumed that the sources are in the far field of the array, consequently the radiation impinging on the array is in the form of plane waves, and secondly, the transmission medium is assumed to be isotropic so that the radiation propagates in straight line. Based on these assumptions, the output of any array element can be represented by a time advanced version or time delayed version of the received signal at a reference element as shown in Figure 1.1.

Since the narrow-band signals are assumed to have the same known frequency ω, the received signals at the reference sensor and the second sensor are respectively given by

$$s(t) = u(t)\, exp[\, j(\omega_o t + v(t))] \tag{1.3}$$

$$s(t-\tau) = u(t-\tau)\, exp[\, j(\omega_o\, (t - \tau) + v(t-\tau))] \tag{1.4}$$

where u(t), and v(t) are the amplitude and phase of s(t) respectively. The signal s(t- τ) at the second sensor is delayed by the time required for the plane wave to propagate through $\Delta \sin \theta$, and if c represents the velocity of propagation, then this time delay τ is given by

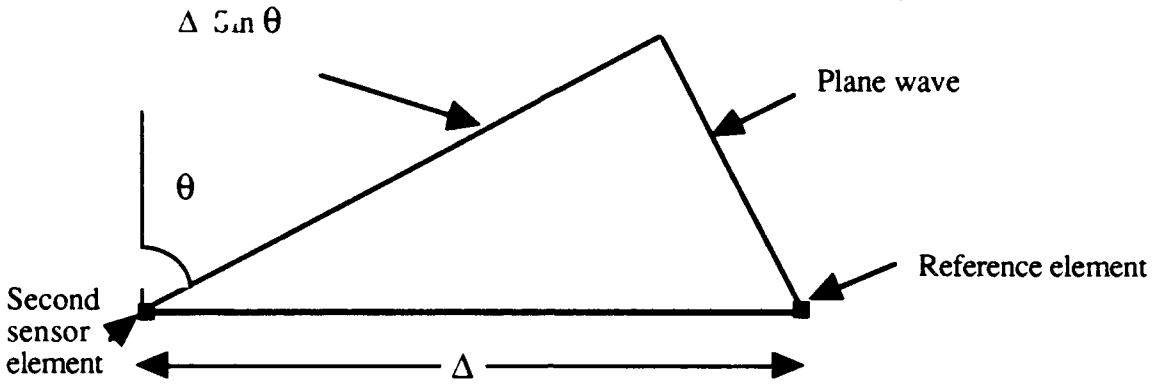


Figure 1.1: A two sensor array

$$\tau = \frac{\Delta \sin \theta}{c} \tag{1.5}$$

The narrow band assumption implies that u(t) and v(t) are slowly varying functions, thus:

$$u(t) = u(t-\tau) \tag{1.6}$$

$$v(t) = v(t-\tau) \tag{1.7}$$

for all possible propagation delays. Thus the effect of a time delay on the received signal is simply a phase shift:

$$s(t-\tau)= s(t)\exp(-j\omega_o\tau) \tag{1.8}$$

Now consider an array consisting of m sensors and receiving signals from d sources located at directions $\theta_1\ \theta_2, \ldots\ \theta_d$ with respect to the line of array, as shown in Figure 1.2.

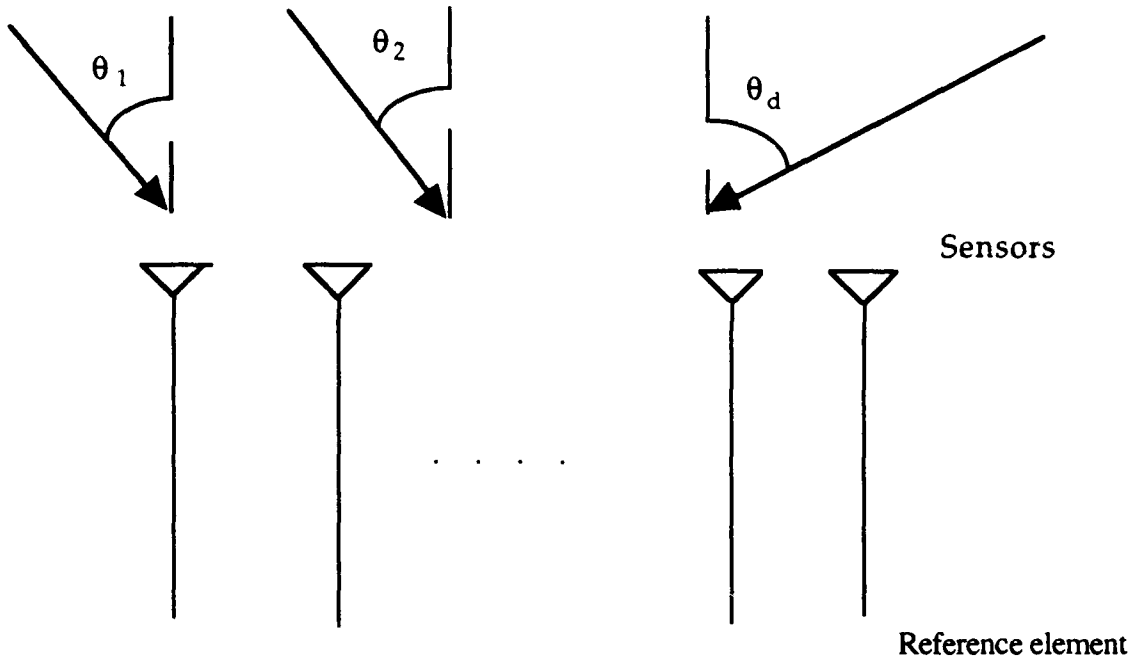


Figure 1.2: Typical array scene.

It is assumed that none of the signals are coherent. Using superposition of signal contribution, the received signal at the k^{th} sensor can be written as

$$x_k(t) = \sum_{i=1}^{d} a_k(\theta_i)\, s_i(t-\tau_k(\theta_i)) + n_k(t)$$

$$= \sum_{i=1}^{d} a_k(\theta_i)\, s_i(t) \exp(-j\omega_o \tau_k(\theta_i)) + n_k(t) \qquad (1.9)$$

where $\tau_k(\theta_i)$ is the propagation delay between a reference point and the k^{th} sensor for the i^{th} wavefront impinging on the array from direction θ_i, $a_k(\theta_i)$ is the corresponding sensor element complex response (gain and phase) at frequency ω_o and $n_k(t)$ stands for the additive noise at the k^{th} sensor. If we let

$$\mathbf{a}(\theta_i) = \{ a_1(\theta_i)\exp(j\omega_o \tau_1(\theta_i)) \ldots a_m(\theta_i)\exp(jw_o \tau_m(\theta_i))\}^H \qquad (1.10)$$

Where H denotes complex conjugate transpose
and $n(t) = \{ n_1(t), n_2(t) \ldots\ldots\ldots\ldots n_m(t)\}^T$

the data model representing the outputs of m sensors becomes

$$\mathbf{x}(t) = \sum_{i=1}^{d} \mathbf{a}(\theta_i) s_i(t) + \mathbf{n}(t) \qquad (1.11)$$

Now by setting

$$\mathbf{A}(\theta) = (\mathbf{a}(\theta_1), \mathbf{a}(\theta_2), \ldots, \mathbf{a}(\theta_d)) \qquad (1.12)$$

and

$$\mathbf{s}(t) = (s_1(t), s_2(t), \ldots, s_d(t))^T \qquad (1.13)$$

x(t) can be rewritten as

$$\mathbf{x}(t) = \mathbf{A}(\theta)\, \mathbf{s}(t) + \mathbf{n}(t) \qquad (1.14)$$

where

$\mathbf{x}(t)$, $\mathbf{n}(t) \in C^m$, $\mathbf{s}(t) \in C^d$ and $\mathbf{A}(\theta) \in C^{m \times d}$ (C: complex plane)

$\mathbf{A}(\theta)$ is called the direction matrix. The columns of $\mathbf{A}(\theta)$ are elements of a set, termed the array manifold, composed of all array response vectors obtained as ranges over the entire space. If signals and noise are assumed to be stationary, zero mean, uncorrelated random processes and further the noises in different sensors are uncorrelated, the spatial correlation matrix of the observed signal vector x(t) is defined by:

$$\mathbf{R}_{xx} = \epsilon\, (\mathbf{x}(t)\, \mathbf{x}^H(t)) \tag{1.15}$$

where ϵ is the expectation operator.

The substitution of Equation (1.14) into (1.15) gives

$$\begin{aligned} \mathbf{R}_{xx} &= \epsilon\, (\, \mathbf{A}(\theta)\, \mathbf{s}(t)\, \mathbf{s}(t)^H (\mathbf{A}(\theta))^H + \sigma^2 . \mathbf{I} \\ &= \mathbf{A}(\theta)\ \mathbf{R}_{ss}\, \mathbf{A}(\theta)^H + \sigma^2 \mathbf{I} \end{aligned} \tag{1.16}$$

where

$$\mathbf{R}_{ss} = \epsilon(\mathbf{s}(t)\mathbf{s}(t)^H) \tag{1.17}$$

and $\sigma^2 . \mathbf{I}$ is the spatial correlation matrix of the noise vector n(t), σ^2 denotes the variance of the elemental noise $n_i(t)$, i = 1, ... n.

1.3 MULTIPLE SIGNAL CLASSIFICATION (MUSIC) ALGORITHM

Consider first the noise free case where

$$x(t) = \sum_{i=1}^{d} \mathbf{a}(\theta_i)\, s_i(t) \qquad (1.18)$$

This means that x(t) is a linear combination of the d steering column vectors of $\mathbf{A}(\theta)$ and is therefore constrained to the d-dimensional subspace of $C_{m,}$, termed the signal subspace, that is spanned by the d columns vectors of $\mathbf{A}(\theta)$. In this case the signal subspace intersects the array manifold at the d steering vectors $\mathbf{a}(\theta_i)$ as shown in Figure 1.3.

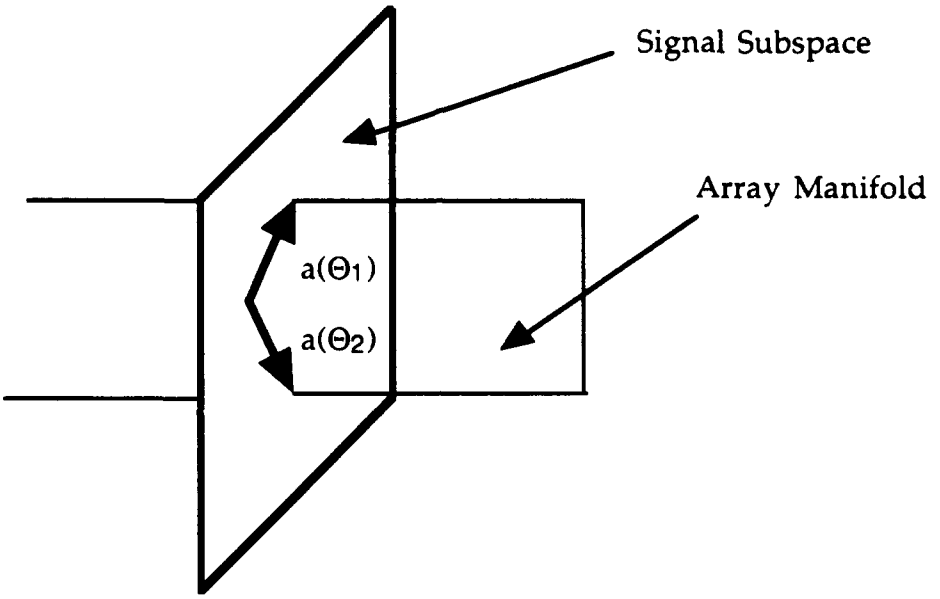


Figure 1.3: Signal subspace and array manifold for a two-source example.

However, when the data is corrupted by noise, the signal subspace has to be estimated and consequently it is expected that the signal subspace will not intersect the array manifold, so the steering vectors closest to the signal subspace will be chosen instead [6]. In the following, it is shown that one set of d independent vectors that span the signal subspace is given by the d eigenvectors corresponding to the d largest eigenvalues of the data covariance matrix. The data covariance matrix

is assumed to be positive definite and Hermetian and consequently its eigendecomposition is given by

$$\mathbf{R}_{xx} = \mathbf{E}\,\Lambda\,\mathbf{E}^{H} \qquad (\mathbf{E}\,\mathbf{E}^{H} = \mathbf{I})$$

$$\Rightarrow \quad \mathbf{R}_{xx}\,\mathbf{E} = \mathbf{E}\,\Lambda$$

$$\Rightarrow \quad (\mathbf{A}(\theta)\mathbf{R}_{ss}\mathbf{A}(\theta)^{H} + \sigma^2.\,\mathbf{I})\,\mathbf{E} = \mathbf{E}\,\Lambda$$

$$\Rightarrow \quad \mathbf{A}(\theta)\mathbf{R}_{ss}\,\mathbf{A}(\theta)^{H}\,\mathbf{E} = \mathbf{E}\,\Lambda - \sigma^2.\,\mathbf{E}$$

$$\Rightarrow \quad \mathbf{E}^{H}\,\mathbf{A}(\theta)\mathbf{R}_{ss}\,\mathbf{A}(\theta)^{H}\,\mathbf{E} = \mathbf{E}^{H}\,\mathbf{E}\,\Lambda - \sigma^2.\,\mathbf{E}^{H}\mathbf{E}$$

$$= \Lambda - \sigma^2.\mathbf{I}$$

$$\Rightarrow \quad \mathbf{A}(\theta)\mathbf{R}_{ss}\,\mathbf{A}(\theta)^{H} = \mathbf{E}(\Lambda - \sigma^2.\mathbf{I})\,\mathbf{E}^{H} \qquad (1.19)$$

Thus the eigenvalues of $\mathbf{A}(\theta)\,\mathbf{R}_{ss}\,\mathbf{A}(\theta)^{H}$ are the d largest eigenvalues of $\mathbf{R}_{xx}$ augmented by σ^2. Also the (m-d) smallest eigenvalues are all equal to σ^2. Now if $(\lambda_i, \mathbf{e}_i)$ is an eigenpair of $\mathbf{R}_{xx}$, then

$$\mathbf{R}_{xx}\,\mathbf{e}_i = \lambda_i\,\mathbf{e}_i \qquad (1.20)$$

and for any i > d,

$$(\mathbf{A}(\theta)\,\mathbf{R}_{ss}\,\mathbf{A}(\theta)^{H} + \sigma^2.\,\mathbf{I})\,\mathbf{e}_i = \sigma^2\mathbf{e}_i$$

$$\Rightarrow \quad \mathbf{A}(\theta)\mathbf{R}_{ss}\,\mathbf{A}(\theta)^{H}\,\mathbf{e}_i = 0 \qquad (1.21)$$

Now from the fact that $\mathbf{A}(\theta)$ and $\mathbf{R}_{ss}$ must have at least one nonsingular submatrix of order d and without loss of generality, suppose that this submatrix consists of the first d rows of $\mathbf{A}_1(\theta)\,\mathbf{R}_{ss}$. Partition $\mathbf{A}_1(\theta)\mathbf{R}_{ss}$ as:

$$\mathbf{A}(\theta)\,\mathbf{R}_{ss} = (\mathbf{A}_1(\theta)\,\mathbf{R}_{ss}\,,\,\mathbf{A}_2(\theta)\,\mathbf{R}_{ss})^{T} \qquad (1.22)$$

The substitution of (1.22) into (1.21) yields

$$\mathbf{A}_1(\theta)\,\mathbf{R}_{ss}\,\mathbf{A}(\theta)^H\,\mathbf{e}_i = 0 \tag{1.23}$$

and

$$\mathbf{A}_2(\theta)\,\mathbf{R}_{ss}\,\mathbf{A}(\theta)^H\,\mathbf{e}_i = 0 \tag{1.24}$$

For the equation (1.23) to be satisfied,

$$\mathbf{A}(\theta)^H\,\mathbf{e}_i = 0 \qquad i > d \tag{1.25}$$

Thus $\mathbf{e}_1, \mathbf{e}_2, \ldots \mathbf{e}_d$ span the same subspace as spanned by the column vectors of $\mathbf{A}(\theta)$. In most situations, the covariance matrices are not known exactly but need to be estimated. Therefore, one can expect that there is no intersection between the array manifold and the signal subspace. However, elements of the array manifold closest to the signal subspace should be considered as potential solution. After determining the number of sources [7], Smith [5] proposed the following function as one possible measure of closeness of an element of the array manifold to the signal subspace

$$P_m(\theta) = \frac{1}{\mathbf{a}^H(\theta)\,\mathbf{E}_n\mathbf{E}_n^H\,\mathbf{a}(\theta)} \tag{1.26}$$

where $\mathbf{E}_n = [\,\mathbf{e}_{d+1}, \mathbf{e}_{d+2}, \ldots, \mathbf{e}_m\,]$

The dominant d peaks over $\theta \in [-\pi, \pi]$ are the desired estimates of the directions of arrival.

For the particular case where the array consists of m sensors uniformly spaced, and if the reference point is taken at the first element of the array, $P_m(\theta)$ is obtained by first calculating the DFT of the vectors spanning the null space of $\mathbf{A}(\theta)\, R_{ss}\, \mathbf{A}(\theta)^H$ or

$$\mathbf{E}_n = [\, \mathbf{e}_{d+1}, \mathbf{e}_{d+2}, \ldots, \mathbf{e}_m] \tag{1.27}$$

If Δ is the distance separating two sensors of the array, an element of the array manifold is given by

$$\mathbf{a}(\theta) = (1, \exp(j2\pi\Delta \sin\theta / \lambda), \ldots, \exp(j2\pi(m-1)\Delta \sin\theta / \lambda))^T \tag{1.28}$$

and the DFT of the vector $\mathbf{e}_i$, $i > d$ is given by

$$F_i = \mathbf{a}^*(\theta)\, \mathbf{e}_i = \sum_{k=1}^{m} \mathbf{e}_{ki} \exp(-j2\pi(k-1)\Delta \sin\theta / \lambda) \tag{1.29}$$

thus

$$P_m(\theta) = \frac{1}{\sum_{i=d+1}^{m} |F_i|^2} \tag{1.30}$$

Summary of the MUSIC algorithm

1) Estimate the data covariance matrix **R**.

2) Perform the eigendecomposition of **R**.

3) Estimate the number of sources.

4) Evaluate $P_m(\theta)$.

5) Find the d largest peaks of $P_m(\theta)$ to obtain estimates of the parameters

Although MUSIC is a high resolution algorithm, it has several drawbacks including the fact that complete knowledge of the array manifold is required, and that is computationaly very expensive as it requires a lot of computations to find the intersection between the array manifold and the signal subspace. In the next section, another algorithm known as ESPRIT will be discussed. Even though it is similar to the MUSIC and exploits the underlying data model but it eliminates the requirement of a time-consuming parameter search.

1.4 ESTIMATION OF SIGNAL PARAMETERS VIA ROTATIONAL INVARIANCE (ESPRIT)

Consider a planar array composed of m pairs of pair identical sensors (doublets) as shown in the Figure 1.4. The displacement between two sensors in each doublet is constant, but the sensor characteristics are unknown [5]. The signal received at the i^{th} doublet can be expressed as

$$x_k(t)= \sum_{i=1}^{d} a_k(\theta_i)\, s_i(t)+ n_{xk}(t)$$

$$y_k(t)= \sum_{i=1}^{d} a_k(\theta_i) \exp(j\omega_o \Delta \sin \theta_i / c)\, s_i(t) + n_{yk}(t) \qquad (1.31)$$

where θ_i is the direction of arrival of the i^{th} source relative to the direction of translational displacement vector. Employing vector notation as in the case of MUSIC, the data vector can be expressed as:

$$\mathbf{x}(t) = \mathbf{A}(\theta)\, \mathbf{s}(t) + \mathbf{n}_x(t)$$

$$\mathbf{y}(t) = \mathbf{A}(\theta)\Phi\, \mathbf{s}(t) + \mathbf{n}_y(t) \qquad (1.32)$$

where

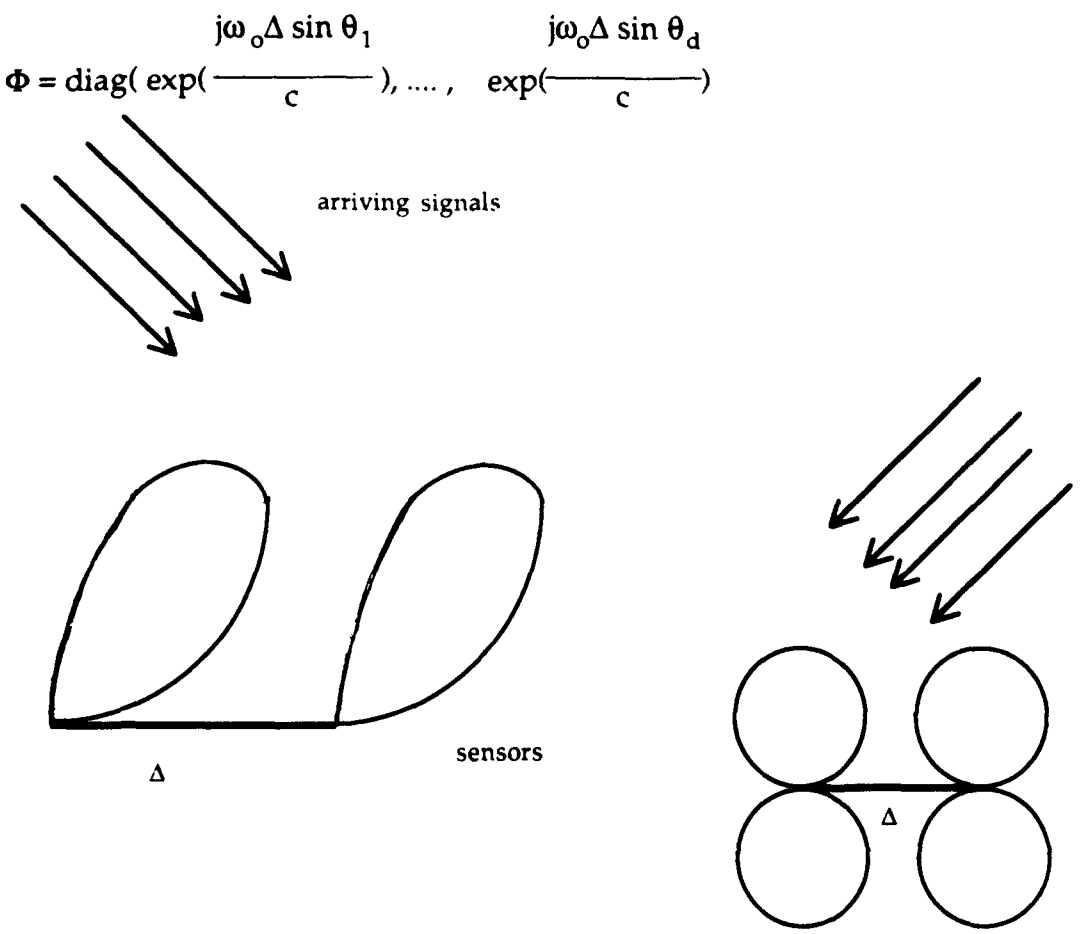


Figure 1.4: Sensor array for ESPRIT.

Now, consider the matrices

$$\mathbf{C}_{xx} = \mathbf{R}_{xx} - \sigma^2 \mathbf{I}$$

$$= \mathbf{A}(\theta)\, \mathbf{R}_{ss}\, \mathbf{A}^*(\theta) \qquad (1.33)$$

and

$$\mathbf{R}_{xy} = \mathbf{A}(\theta)\, \mathbf{R}_{ss}\, \Phi^* \mathbf{A}^*(\theta) \qquad (1.34)$$

In the computation of $\mathbf{R}_{xx}$ the noise in different sensors is assumed to be uncorrelated ($\mathcal{E}[n_x(t)\, n_y(t)] = 0$).

The eigenvalues of the matrix pencil ($\mathbf{C}_{xx}$, $\mathbf{R}_{xy}$) are obtained by solving

$$\left| \mathbf{C}_{xx} - \gamma \mathbf{R}_{xy} \right| = 0 \tag{1.35a}$$

or

$$\left| \mathbf{A}(\theta)\, \mathbf{R}_{ss} (\mathbf{I} - \gamma \Phi^{*}) \mathbf{A}^{*}(\theta) \right| = 0 \tag{1.36b}$$

Now from the fact that $\mathbf{A}(\theta)$ and $\mathbf{R}_{ss}$ are full rank matrices, Equation (1.35b) reduces to

$$\left| \mathbf{I} - \gamma \Phi^{*} \right| = 0 \tag{1.36}$$

and the desired singular values are

$$\gamma_{k} = \exp\left(\frac{j\omega_{o}\Delta \sin\theta_{k}}{c}\right) \qquad k=1,\ldots,d \tag{1.37}$$

Thus the direction of arrival can be obtained without involving a search technique as in the MUSIC case, and in that respect computation and storage costs are reduced considerably. Also it can be concluded that the generalized eigenvalue matrix associated with the matrix pencil ($\mathbf{C}_{xx}$, $\mathbf{R}_{xy}$) is given by:

$$\mathbf{A} = \begin{vmatrix} \Phi & 0 \\ 0 & 0 \end{vmatrix} \tag{1.38}$$

However, due to error in estimating $\mathbf{R}_{xx}$ and $\mathbf{R}_{xy}$ from a finite data sample as well as round-off errors introduced during the squaring of the data, the relation

between $\mathbf{A}$ and Φ given above is not exactly satisfied, which make this method suboptimal.

The following procedure is proposed to estimate the generalized eigenvalues [7]

1) Find the data covariance matrix of the complete 2m sensors, denoted by $\mathbf{R}_{zz}$.

2) Estimate the number of sources d.

3) Estimate the noise variance (average of the 2m - d noise eigenvalues).

4) Compute $\mathbf{R}_z - \sigma^2 \mathrm{I}$, then $\mathbf{A}(\theta)\,\mathbf{R}_{ss}\,\mathbf{A}^*(\theta)$ and $\mathbf{A}(\theta)\,\mathbf{R}_{ss}\,\Phi^*\mathbf{A}^*(\theta)$ are then the top left and top right blocks.

5) Calculate the generalized eigenvalues of the matrix pencil $(\mathbf{C}_{xx}, \mathbf{R}_{xy})$ and choose the d ones that lie close to the unit circle.

1.5 TOTAL LEAST SQUARE (TLS) ESPRIT

The last method is based on having a very good estimate of the noise variance, a condition difficult to satisfy in most real cases. This may yield overall inferior results. To circumvent this difficulty to some extent, the total-least-square (TLS ESPRIT) scheme is used instead.

Let

$$z(t)=\begin{vmatrix} x(t) \\ y(t) \end{vmatrix} = \overline{\mathbf{A}}\, s(t) + \mathbf{n}_z(t) \tag{1.39}$$

where

$$\overline{\mathbf{A}} = \begin{vmatrix} \mathbf{A} \\ \mathbf{A}\Phi \end{vmatrix}, \quad \mathbf{n}_z(t) = \begin{vmatrix} \mathbf{n}_x(t) \\ \mathbf{n}_y(t) \end{vmatrix} \tag{1.40}$$

and let $\mathbf{E}_s = [\mathbf{e}_1, \mathbf{e}_2, \ldots \mathbf{e}_d]$ be the $(2m \times d)$ matrix composed of the eigenvectors corresponding to the d largest eigenvalues of $(\mathbf{R}_{zz}, \mathbf{I})$. Since the columns of $\mathbf{E}_s$ and $\overline{\mathbf{A}}$ span the same subspace, then there must exist a non-signular Γ matrix of dimension d, such that

$$\mathbf{E}_s = \overline{\mathbf{A}}\ \Gamma \tag{1.41}$$

Now define two $m \times d$ matrices $\mathbf{E}_x$ and $\mathbf{E}_y$ by partitioning $\mathbf{E}_s$ as

$$\mathbf{E}_s = \begin{vmatrix} \mathbf{E}_x \\ \mathbf{E}_y \end{vmatrix} = \begin{vmatrix} \mathbf{A}\Gamma \\ \mathbf{A}\Phi\Gamma \end{vmatrix} \tag{1.42}$$

Since $\mathbf{E}_x$ and $\mathbf{E}_y$ share a common space (i.e. the columns of both $\mathbf{E}_x$ and $\mathbf{E}_y$ are a linear combination of the columns of A), then the rank of $\mathbf{E}_{xy} = [\mathbf{E}_x \mid \mathbf{E}_y]$ is d which implies that there exist a unique $2d \times d$ matrix F of rank d such that

$$0 = [\mathbf{E}_x \mid \mathbf{E}_y]\ \mathbf{F} = \mathbf{E}_x\mathbf{F}_x + \mathbf{E}_y\mathbf{F}_y$$

$$= \mathbf{A}\ \Gamma\ \mathbf{F}_x + \mathbf{A}\Phi\ \Gamma\ \mathbf{F}_y \tag{1.43}$$

(F span the null-space of $[\mathbf{E}_x \mid \mathbf{E}_y]$).

In the above equation $[\mathbf{E}_x \mid \mathbf{E}_y]$ is an $m \times 2d$ matrix, it can be seen as consisting of m vectors in a 2d dimensional space, and the set of all vectors which transform into the zero vector (i.e. which satisfy $[\mathbf{E}_x \mid \mathbf{E}_y]\ x = 0$) is called the null space of A, and it has a dimension, 2d-rank $[\mathbf{E}_x \mid \mathbf{E}_y]$, or d. Now if

$$\Psi = -\mathbf{F}_x[\mathbf{F}_y^{-1}] \tag{1.44}$$

then

$$\mathbf{A}\,\Gamma\,\Psi\,\Gamma^{-1} = \mathbf{A}\Phi \tag{1.45}$$

If A is assumed to be a full rank matrix, then

$$\Gamma\,\Psi\,\Gamma^{-1} = \Phi \tag{1.46}$$

Thus the eigenvalues of Ψ correspond to the diagonal element of Φ.

Summary of the TLS ESPRIT

1) Obtain an estimate of the data covariance matrix $\mathbf{R}_{zz}$, denoted by $\overline{\mathbf{R}}_{zz}$.

2) Perform the eigendecomposition of $\overline{\mathbf{R}}_{zz}$ as $\overline{\mathbf{R}}_{zz} = \mathbf{E}\,\Lambda\,\mathbf{E}$

3) Estimate the number of sources d.

4) Obtain $\mathbf{E}_s = [\,\mathbf{e}_1, \mathbf{e}_2, \ldots \mathbf{e}_d\,]$ and decompose it to

obtain $\begin{vmatrix} \mathbf{E}_x \\ \mathbf{E}_y \end{vmatrix}$

5) Compute the eigendecomposition of $\mathbf{E}_{xy}^H\mathbf{E}_{xy} = \begin{vmatrix} \mathbf{E}_x^H \\ \mathbf{E}_y^H \end{vmatrix}\,[\mathbf{E}_x \mid \mathbf{E}_y] = \mathbf{E}\,\Lambda\,\mathbf{E}$

and partition **E** into four d × d submatrices

$$\mathbf{E} = \begin{vmatrix} \mathbf{E}_{11}\,\mathbf{E}_{12} \\ \mathbf{E}_{21}\,\mathbf{E}_{22} \end{vmatrix}$$

6) Calculate the eigenvalues of $\Psi = -\,\mathbf{E}_{12}[\,\mathbf{E}_{22}^{-1}]$

7) Estimate $\theta_k = f^{-1}(\Phi_k)$

$$= \mathrm{Sin}^{-1}\{ c \arg(\phi_k) / (\omega_0 \Delta)\}$$

1.6 IMPROVED TLS ESPRIT

By considering the eigendecomposition of the data matrix $\mathbf{R}_{zz}$ of rank d, following equation can be written.

$$\mathbf{R}_{zz}\mathbf{e}_i = \lambda_i \mathbf{e}_i = \sigma^2 \mathbf{e}_i \qquad i=d+1, ..., 2m \tag{1.47}$$

Using the same procedure as in the MUSIC algorithm

$$\overline{\mathbf{A}}\ \mathbf{G} = 0 \tag{1.48}$$

where

$$\mathbf{G} = [\ \mathbf{e}_{d+1}, \mathbf{e}_{d+2}, \ldots\ \mathbf{e}_{2m}\ ]$$

Now from the fact that $\overline{\mathbf{A}}$ and **G** can be partitioned as

$$\overline{\mathbf{A}} = \begin{vmatrix} \mathbf{A}\Phi \\ \mathbf{A} \end{vmatrix} \quad \text{and} \quad \mathbf{G} = \begin{vmatrix} \mathbf{G}_x \\ \mathbf{G}_y \end{vmatrix} \tag{1.49}$$

Hence

$$(\mathbf{A}^H, \Phi^H \mathbf{A}^H) \begin{vmatrix} \mathbf{G}_x \\ \mathbf{G}_y \end{vmatrix} = 0 \tag{1.50}$$

or

$$\mathbf{A}^H\mathbf{G}_x + \Phi^H\mathbf{A}^H\mathbf{G}_y = 0$$

$$\Rightarrow \quad \mathbf{A}^H\mathbf{G}_x = -\Phi^H\mathbf{A}^H\mathbf{G}_y$$

$$\Rightarrow \quad \mathbf{G}_x^H\mathbf{A} = -\mathbf{G}_y^H\mathbf{A}\Phi \tag{1.51}$$

By multiplying both sides of the above equation by $\mathbf{T}$ defined in Equation(1.41).

$$\mathbf{G}_x^H\mathbf{A}\,\mathbf{T} = -\mathbf{G}_y^H\mathbf{A}\Phi\,\mathbf{T} \tag{1.51a}$$

or

$$\mathbf{G}_x^H\mathbf{E}_x = -\mathbf{G}_y^H\mathbf{E}_y \tag{1.51b}$$

Because $\mathbf{E}_x$ and $\mathbf{E}_y$ span the same subspace, then the objective in the previous TLS algorithm is to find a matrix $\psi \in \mathbf{C}^{d\times d}$ such that

$$\mathbf{E}_x\psi = \mathbf{E}_y. \tag{1.52}$$

The substitution of (1.52) into (1.51b) yields

$$\mathbf{G}_x^H\mathbf{E}_x = -\mathbf{G}_y^H\mathbf{E}_x\psi \tag{1.53}$$

Thus if there exist ψ which transforms $\mathbf{E}_x$ into $\mathbf{E}_y$, this transformation must also transform $-\mathbf{G}_y^H\mathbf{E}_x$ into $\mathbf{G}_x^H\mathbf{E}_x$ (Note that $-\mathbf{G}_y^H\mathbf{E}_x$ and $\mathbf{G}_x^H\mathbf{E}_x$ span the same subspace as spanned by the columns of $\mathbf{E}_x$ or $\mathbf{E}_y$).

In practical situations, where only a finite number of noisy measurements are available, Equations (1.51) and (1 53) can not be satisfied exactly. A criterion for obtaining a suitable estimate of ψ must be formulated. The TLS is a method of fitting that is appropriate in this case because $\mathbf{E}_x$, $\mathbf{E}_y$, $\mathbf{G}^H{}_y\mathbf{E}_x$, and $\mathbf{G}^H{}_x\mathbf{E}_x$ are all noisy measurements .

To find a common transformation which satisfies both (1.51) and (1.53) , define

$$\mathbf{H}_1 = \begin{vmatrix} \mathbf{E}_x \\ -\mathbf{G}_y^H \mathbf{E}_x \end{vmatrix} \quad \text{and} \quad \mathbf{H}_2 = \begin{vmatrix} \mathbf{E}_y \\ \mathbf{G}_x^H \mathbf{E}_x \end{vmatrix} \tag{1.54}$$

thus ψ is given by

$$\mathbf{H}_1 \psi = \mathbf{H}_2 \tag{1.55}$$

The previous TLS algorithm applied to the model $\mathbf{E}_x \psi = \mathbf{E}_y$ can be viewed as using m observations (the number of rows of $\mathbf{E}_x$ or $\mathbf{E}_y$). By using Equation (1.55), it is easily verified that the number of observations is increased from m to 3m- d . Thus a better estimate of ψ is believed to be achieved, and the algorithm of the improved TLS will be the same as for the TLS with the exception of replacing $\mathbf{E}_{xy}$ by $\mathbf{E}_{H_1 H_2}$.

However the same solution for Ψ can be achieved by considering instead the d matrices

$$\mathbf{K}_i = \begin{vmatrix} \mathbf{H}_1^H \mathbf{H}_1 & \mathbf{H}_1^H \mathbf{h}_{2i} \\ \mathbf{h}_{2i}^H \mathbf{H}_1 & \mathbf{h}_{2i}^H \mathbf{h}_{2i} \end{vmatrix} \tag{1.56}$$

where $\mathbf{h}_{2i}$ is the i^{th} column of the matrix $\mathbf{H}_2$. If λ_{d+1} is the smallest eigenvalue of $\mathbf{K}_i$, then

$$\mathbf{K}_i \mathbf{e}_{d+1} = \lambda_{d+1} \mathbf{e}_{d+1} \tag{1.57}$$

Now by transforming $\mathbf{e}_{d+1}$ into $\begin{vmatrix} X_i \\ -1 \end{vmatrix}$, X_i solves the TLS problem and gives the i^{th} column of Ψ [8].

This transformation is very useful for parallel processing as it avoids the computation of $\mathbf{F}_y^{-1}$ given in (1.44) to find Ψ. However this method has a disadvantage as the eigendecomposition of d matrices must be performed at the same time.

1.7 CONCLUSIONS

A theorital background for MUSIC and ESPRIT algorithmn for DOA estimation has been presented. An improved ESPRIT algorithm is also given which improves estimation of DOA's. As seen above for both MUSIC and ESPRIT algorithm the number of sensors is equal are dependent on the number of source whose direction of arrival has to be estimated. It is considered that the number of source never exceed seven, and hence number of sensors is always considered to be eight from next chapter onwards.

Chapter 2

EIGENDECOMPSITION USING HOUSEHOLDER'S TRANSFORMATION AND GIVEN'S ROTATION

1.1 SYMMETRICAL EIGEN DECOMPOSITION PROBLEM

It is well known that the symmetric eigendecomposition problem is one of the fundamental problems in signal processing as it arises in many applications such as DOA's estimation and spectral estimation. Most methods reduce the problem to the generalized eigendecomposition problem by computing the data covariance matrix. Householder's method is a technique used for reducing the bandwidth of the data covariance matrix by transforming it to a tridiagonal one under congruent transformations without affecting the values of the eigenvalues [9]. In fact, if (x,λ) is an eigenpair of the covariance data matrix $\mathbf{R}_{xx}$, and if $\mathbf{N}$ is an orthogonal matrix it can be shown that $(\mathbf{N}^H \mathbf{x}, \lambda)$ is an eigenpair of $\mathbf{N}^H\mathbf{R}_{xx}\mathbf{N}$.

In order to transform the m x m data covariance matrix $\mathbf{R}_{xx}$ to a tridiagonal matrix, T, , m-2 Householder' s transformations (N_i, i=1,2....m-2) are determined such that $\mathbf{N}^H \mathbf{R}_{xx}\mathbf{N} = \mathbf{T}$, where

$$\mathbf{N} = \mathbf{N}_1 \cdot \mathbf{N}_2 \cdots \mathbf{N}_{m-2} \cdot$$

Each transformation is determined to eliminate a whole row and column above and below the subdiagonals without disturbing any previously zeroed rows and columns. The basic iteration sequence of operation for this transformation method can be stated as

$$\mathbf{R}_1 = \mathbf{R}_{xx} \qquad (2.1)$$

$$\mathbf{U}_1 = \mathbf{I} \qquad (2.2)$$

begin

For k=1,2,...,m-2,

$$\mathbf{R}_{k+1} = \mathbf{N}_k^H \mathbf{R}_k \mathbf{N}_k \tag{2.3}$$

$$\mathbf{U}_{k+1} = \mathbf{N}_k^H \mathbf{U}_k \tag{2.4}$$

end

$$\mathbf{T} = \mathbf{R}_{m-1} \tag{2.5}$$

$$\mathbf{U} = \mathbf{U}_{m-1} \tag{2.6}$$

T is a tridiagonal matrix and **U** is an orthogonal matrix of eigenvectors [$\mathbf{u}_1$, $\mathbf{u}_2$,...,$\mathbf{u}_m$] which can be related to the matrix of eigenvectors **X** of the original problem by

$$\mathbf{X} = \prod_{k=1}^{m-2} \mathbf{N}_k \, \mathbf{U} \tag{2.7}$$

where

$$\mathbf{N}_k = \begin{array}{c} \left[\begin{array}{c|c} \mathbf{I} & \mathbf{0} \\ \hline \mathbf{0} & \overline{\mathbf{N}_k} \end{array}\right] \begin{array}{l} k \\ m-k \end{array} \\ \begin{array}{cc} k & m-k \end{array} \end{array} \tag{2.8}$$

and

$$\overline{N}_k = I - \frac{2 w w^H}{w^H w}$$

where w is a vector chosen such that the matrix N_k is orthogonal. The method is best illustrated by carrying out the first reduction. For k=1, the transformation can be written as

$$R_2 = N_1^H R_1 N_1$$

where

$$R_1 = \begin{array}{c} \begin{matrix} 1 \end{matrix} \\ \begin{matrix} 1 \end{matrix} \left[\begin{array}{c|c} r_{11} & r_1^H \\ \hline r_1 & \overline{R}_1 \end{array}\right] \begin{matrix} 1 \\ m-1 \end{matrix} \\ \begin{matrix} 1 & m-1 \end{matrix} \end{array}$$

Therefore

$$R_2 = \left[\begin{array}{c|c} 1 & 0 \\ \hline 0 & \overline{N}_1^H \end{array}\right] \left[\begin{array}{c|c} r_{11} & r_1^H \\ \hline r_1 & \overline{R}_1 \end{array}\right] \left[\begin{array}{c|c} 1 & 0 \\ \hline 0 & \overline{N}_1 \end{array}\right]$$

$$= \left[\begin{array}{c|c} r_{11} & r_1^H \\ \hline \overline{N}_1^H r_1 & \overline{N}_1^H \overline{R}_1 \end{array}\right] \left[\begin{array}{c|c} 1 & 0 \\ \hline 0 & \overline{N}_1 \end{array}\right]$$

$$= \begin{matrix} 1 \\ 1 \\ \\ \\ 1 \end{matrix} \left[\begin{array}{c|c} r_{11} & \mathbf{r}_1^H \bar{N}_1 \\ \hline \bar{N}_1^H \mathbf{r}_1 & \bar{N}_1^H \bar{R}_1 \bar{N}_1 \end{array}\right] \begin{matrix} 1 \\ \\ \\ \\ m-1 \end{matrix}$$

$$\begin{matrix} & 1 & & m-1 \end{matrix}$$

In order for the first term $\bar{N}_1^H \mathbf{r}_1$ to be null except for the first element, 'w' should have the following form

$$\mathbf{w} = \mathbf{r}_1 + \beta \mathbf{e}_1$$

where $\mathbf{e}_1 = (1, 0, 0, 0...)^T$

and $\mathbf{r}_1 = (r_{21}, r_{31}, ... r_{n1})^T$

β is a complex number that is to be determined such that

$$N)_1^H \mathbf{r}_1 = \left\{ I - \frac{2\mathbf{w}\mathbf{w}^H}{\mathbf{w}^H\mathbf{w}} \right\}^H \mathbf{r}_1$$

$$= \left\{ I - \frac{2\mathbf{w}\mathbf{w}^H}{\mathbf{w}^H\mathbf{w}} \right\} \mathbf{r}_1$$

$$= \mathbf{r}_1 - \frac{2(\mathbf{r}_1 + \beta \mathbf{e}_1)(\mathbf{r}_1^H + \beta^* \mathbf{e}_1^T)\mathbf{r}_1}{(\mathbf{r}_1^H + \beta^* \mathbf{e}_1^T)(\mathbf{r}_1 + \beta \mathbf{e}_1)}$$

$$= -\beta \mathbf{e}_1$$

For this equation to be satisfied, we should have

$$2(\mathbf{r}_1^H + \beta^* \mathbf{e}_1^T)\mathbf{r}_1 = (\mathbf{r}_1^H + \beta^* \mathbf{e}_1^T)(\mathbf{r}_1 + \beta \mathbf{e}_1)$$

or

$$2\mathbf{r}_1^H \mathbf{r}_1 + 2\beta^* \mathbf{e}_1^T \mathbf{r}_1 = \mathbf{r}_1^H \mathbf{r}_1 + \beta^* \mathbf{e}_1^T \mathbf{r}_1 + \beta \mathbf{r}_1^H \mathbf{e}_1 + \beta^* \beta$$

One solution is given by

$$\mathbf{r}_1^H \mathbf{r}_1 = \beta^* \beta \tag{2.9}$$

and

$$\beta^* \mathbf{e}_1^T \mathbf{r}_1 = \beta \mathbf{r}_1^H \mathbf{e}_1) \tag{2.10}$$

multiplying both sides of (2.10) by β^*.

$$(\beta^*)^2 \mathbf{e}_1^T \mathbf{r}_1 = \beta^* \beta \mathbf{r}_1^H \mathbf{e}_1 \tag{2.11}$$

Substitution of (2.9) in (2.11) yields

$$(\beta^*)^2 = \frac{\mathbf{r}_1^H \mathbf{r}_1 \; \mathbf{r}_1^H \mathbf{e}_1}{\mathbf{e}_1^T \mathbf{r}_1}$$

Let

$$\mathbf{r}_1^H \mathbf{r}_1 = \left\| \mathbf{r}_1 \right\|^2 = \beta_1^2$$

and from the fact that

$$\mathbf{r}_1^H \mathbf{e}_1 = \begin{bmatrix} r^*_{21} & r^*_{31} \cdots\cdots & r^*_{21} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} = r^*_{21}$$

$$e_1^T r_1 = \begin{bmatrix} 1 & 0 ..... & 0 \end{bmatrix} \begin{bmatrix} r_{21} \\ r_{31} \\ . \\ . \\ . \\ r_{m1} \end{bmatrix} = r_{21}$$

we get

$$(\beta^*)^2 = \beta_1^2 \frac{r^*_{21}}{r_{21}}$$

$$= \beta_1^2 \frac{(r^*_{21})^2}{r^*_{21} \cdot r_{21}}$$

$$= \beta_1^2 \frac{(r^*_{21})^2}{|r_{21}|^2}$$

or

$$\beta = \beta_1 \frac{r_{21}}{|r_{21}|}$$

By choosing β given by the previous equation, the first column of $\mathbf{R}_2$ has the form $(r_{11}, -\beta, 0, 0, 0...0)^T$, and because of the Hermetian property,the first row becomes $(r_{11}, -\beta^*, 0, 0,...0)$.

In the following a method is given for a recursive application of this result to calculate the elements of $\mathbf{R}_2, \mathbf{R}_{3,\ldots,} \mathbf{R}_{m-2}$. For instance

$$\mathbf{R}_2 = \mathbf{N}_1^H \mathbf{R}_1 \mathbf{N}_1$$

$$= \left[\mathbf{I} - \frac{2\mathbf{w}\mathbf{w}^H}{\mathbf{w}^H\mathbf{w}}\right] \mathbf{R}_1 \left[\mathbf{I} - \frac{2\mathbf{w}\mathbf{w}^H}{\mathbf{w}^H\mathbf{w}}\right]$$

$$= \left[\mathbf{R}_1 - \frac{2\mathbf{w}\mathbf{w}^H\mathbf{R}_1}{\mathbf{w}^H\mathbf{w}}\right]\left[\mathbf{I} - \frac{2\mathbf{w}\mathbf{w}^H}{\mathbf{w}^H\mathbf{w}}\right]$$

$$= \mathbf{R}_1 - \frac{2\,\mathbf{w}\mathbf{w}^H}{\mathbf{w}^H\mathbf{w}}\mathbf{R}_1 - \mathbf{R}_1\frac{2\,\mathbf{w}\mathbf{w}^H}{\mathbf{w}^H\mathbf{w}} + 4\frac{\mathbf{w}\mathbf{w}^H\mathbf{R}_1\mathbf{w}\mathbf{w}^H}{\mathbf{w}^H\mathbf{w}\,\mathbf{w}^H\mathbf{w}} \qquad (2.12)$$

where $\mathbf{w} = \mathbf{r}_1 + \beta\,\mathbf{e}_1$,

By defining $c = \frac{\mathbf{w}^H\mathbf{w}}{2}$, and $\mathbf{d} = \mathbf{R}_1\mathbf{w}$

Equation (2.12) can be rewritten as follows:

$$\mathbf{R}_2 = \mathbf{R}_1 - \frac{\mathbf{w}\,\mathbf{d}^H}{c} - \frac{\mathbf{d}\,\mathbf{w}^H}{c} + \frac{\mathbf{w}\,(\mathbf{w}^H\mathbf{d})\,\mathbf{w}^H}{c^2}$$

$$= \mathbf{R}_1 - \frac{\mathbf{w}\,\mathbf{d}^H}{c} - \frac{\mathbf{d}\,\mathbf{w}^H}{c} + \frac{\mathbf{w}\,(\mathbf{w}^H\mathbf{d})\,\mathbf{w}^H}{2\,.\,c^2} + \frac{\mathbf{w}\,(\mathbf{w}^H\mathbf{d})\,\mathbf{w}^H}{2\,.\,c^2}$$

$$= \mathbf{R}_1 - \left(\frac{\mathbf{d}}{c} - \frac{\mathbf{w}(\mathbf{w}^H \mathbf{d})}{2 . c^2} \right) \mathbf{w}^H - \mathbf{w} \left(\frac{\mathbf{d}^H}{c} - \frac{(\mathbf{w}^H \mathbf{d}) \mathbf{w}^H}{2 . c^2} \right)$$

Let

$$\mathbf{v} = \frac{\mathbf{d}}{c} - \frac{\mathbf{w}(\mathbf{w}^H \mathbf{d})}{2c^2}$$

then

$$\mathbf{v}^H = \frac{\mathbf{d}^H}{c} - \frac{\mathbf{d}^H \mathbf{w} \mathbf{w}^H}{2 . c^2}$$

Now from the fact that

$$\mathbf{d}^H \mathbf{w} = \mathbf{w}^H \mathbf{d}$$

$\mathbf{v}^H$ can be rewritten as

$$\mathbf{v}^H = \frac{\mathbf{d}^H}{c} - \frac{(\mathbf{w}^H \mathbf{d}) \mathbf{w}^H}{2 . c^2}$$

and Equation (2.12) reduces to

$$\mathbf{R}_2 = \mathbf{R}_1 - \mathbf{w} \mathbf{v}^H - \mathbf{v} \mathbf{w}^H \qquad (2.13)$$

The choice of above equation is primarily motivated by the interest in the application of parallel processing in computing the elements of the matrix $\mathbf{R}_2$; that is, all the columns of $\mathbf{R}_2$ can be computed in parallel as:

$$\mathbf{R}_{2j} = \mathbf{R}_{1j} - v_j^* \mathbf{w} - w_j^* \mathbf{v} \qquad (2.14)$$

j=1,2.....8

where $\mathbf{R}_{2j}$ and $\mathbf{R}_{1j}$ are the j^{th} column of $\mathbf{R}_2$ and $\mathbf{R}_1$, and $\mathbf{v}_j$ and $\mathbf{w}_j^*$ are the j^{th} components of **v** and **w** respectively.

Given the tridiagonal matrix **T** and defining $\mathbf{U} = \mathbf{N}^H$ which is obtained from Householders transformation, the QR algorithm may be used to compute eigenvalues and eigenvectors. This is achieved by producing a sequence of transformations based on orthogonal matrices and illustrated by the following algorithm.

$\mathbf{T}_1 = \mathbf{T}$

$\mathbf{U}_1 = \mathbf{N}^H$

begin

for k=1, n

$\mathbf{R}_k = \mathbf{Q}_k^H \mathbf{T}_k$

$\mathbf{T}_{k+1} = \mathbf{R}_k \mathbf{Q}_k$

$\mathbf{U}_{k+1} = \mathbf{Q}_K^H \mathbf{U}_k$

end

$\Sigma = \mathbf{T}_{n+1}$

$\mathbf{X}^H = \mathbf{U}_{n+1}$

After k iterations **T** will be approximately a diagonal matrix Σ whose diagonal elements approximate the eigenvalues of the original matrix, and the appropriate

eigenvectors are given by the columns of the matrix $\mathbf{X}$. The orthogonal matrices $\mathbf{Q}_k$ in the QR algorithm are the product of m-1 rotations $\mathbf{Q}_{(k,i)}$; i=1,...,m-1, in the (i,i+1) planes respectively. Each rotation $\mathbf{Q}^H_{(k,i)}$ is defined as a matrix which is an identity matrix except for the entries (i,i), (i,i+1), (i+1,i), and (i+1, i+1) which together form a 2 x 2 matrix given by

$$\mathbf{Q}^H_{(k,i)} = \begin{vmatrix} c & s^* \\ -s & c \end{vmatrix} \tag{2.15}$$

The factorization producing $\mathbf{R}_k$ and $\mathbf{Q}_k$ from the original matrix $\mathbf{T}$ is explained as follows. Each subdiagonal element can be eliminated by a plane rotation, the first one is

$$\mathbf{Q}^H_{(1,1)}\ \mathbf{R}_1 = \begin{vmatrix} 1 & \exp(-j\theta) \\ -\exp(j\theta) & 1 \end{vmatrix} \begin{vmatrix} t_{11} & \cdot \\ t_{21} & \cdot \end{vmatrix} \tag{2.16}$$

The (2,1) entry in this product should be equal to zero, thus

$$-t_{11} \exp(j\theta) + t_{21} = 0 \tag{2.17}$$

or

$$\exp(j\theta) = t_{21}/\ t_{11} = r \exp(j\phi)$$

where $r = |\ t_{21}/\ t_{11}\ |$, and $\phi = \arg(t_{21}) - \arg(t_{11})$

To have a unitary matrix , the matrix $\mathbf{Q}^H_{(1,i)}$ is chosen as

$$\begin{vmatrix} \frac{1}{\sqrt{1+r^2}} & \frac{r\exp(-j\phi)}{\sqrt{1+r^2}} \\ \frac{-r\exp(j\phi)}{\sqrt{1+r^2}} & \frac{1}{\sqrt{1+r^2}} \end{vmatrix} \tag{2.18}$$

For a Hermetian tridiagonal matrix, we have

$$\frac{1}{\sqrt{1+r^2}} = \frac{t_{11}}{\sqrt{t_{11}^2+|t_{21}|^2}} \tag{2.19}$$

and

$$\frac{r\exp(j\phi)}{\sqrt{1+r^2}} = \frac{t_{21}}{\sqrt{t_{11}^2+|t_{21}|^2}} \tag{2.20}$$

and the above matrix reduces to

$$\begin{vmatrix} \frac{t_{11}}{\sqrt{t_{11}^2+|t_{21}|^2}} & \frac{t_{21}^*}{\sqrt{t_{11}^2+|t_{21}|^2}} \\ \frac{-t_{21}}{\sqrt{t_{11}^2+|t_{21}|^2}} & \frac{t_{11}}{\sqrt{t_{11}^2+|t_{21}|^2}} \end{vmatrix} \tag{2.21}$$

In comparing the various methods for solving the eigendecomposition problem, there are numerous factors that one must consider . Perhaps, the primary factor is that of the relative efficiency of the method under consideration. One

criterion commonly used in the eigendecomposition problem for determining the efficiency of a particular method is the time required to solve this problem, and hence one might rely on special purpose hardware and exploit pipelining and parallel processing to achieve high throughput rates. We now turn to the question of what efficient algorithm is to be used to perform the eigendecomposition of a tridiagonal matrix. Phillips and Robertson presented an algorithm for tridiagonal QR[15] which has been modified and incorporated in this work.

Let the entries of the diagonal and subdiagonal elements of the tridiagonal matrix **T**, shown below be a(m,k) and b(m,k) respectively where m is row or column number and k is iteration number, and let c(i,k) and s(i,k) be the entries of the rotations used in rotating rows i and i+1 at the $(k+1)^{th}$ iteration.

$$T = \begin{bmatrix} a(1,k) & b(2,k) & & & \\ b(2,k) & a(2,k) & b(3,k) & & \\ & b(3,k) & a(3,k) & & \\ & & \ddots & \ddots & \\ & & & \ddots & \ddots \end{bmatrix}$$

The updated matrix $\mathbf{T}_{k+1}$ can be expressed as

$$\mathbf{T}_{k+1} = \mathbf{Q}^{H}_{(k,m-1)} \mathbf{Q}^{H}_{(k,m-2)} \cdots \mathbf{Q}^{H}_{(k,1)} \mathbf{T}_k \mathbf{Q}_{(k,1)} \mathbf{Q}_{(k,2)} \cdots \mathbf{Q}_{(k,m-1)} \qquad (2.22)$$

Using the associative property of matrix products, the multiplication of $\mathbf{T}_k$ by

$Q^H_{(k,\ i-2)}\ Q^H_{(k,\ i-3)} \cdots Q^H_{(k,\ 1)}$ from the left results in a matrix R'_k whose subdiagonal entries up to b(i-1,i) are forced to zero , that is

$$T = \begin{bmatrix} x(2,k) & & & & & & \\ & x(3,k) & & & & & \\ & & x(4,k) & & & & \\ & & b(i,k) & a(i,k) & b(i+1,k) & & \\ & & & b(i+1,k) & a(i+1,k) & . & \\ & & & & . & . & . \\ & & & & & . & . \end{bmatrix} \qquad -(2.23)$$

and T_{k+1} can be rewritten as

$$T_{k+1} = Q^H_{(k,m-1)}\, Q^H_{(k,m-2)}\ Q^H_{(k,i-1)}\, R'_k\, Q_{(k,1)}\, Q_{(k,2)} \cdots Q_{(k,m-1)}$$

Now after a little thought, it can be shown that the multiplication of R'_k by $Q^H_{(k,\ i)} Q^H_{(k,\ i-1)}$ and $Q_{(k,\ i-1)} Q_{(k,\ i)}$ from the left and right respectively results in the updated entries b(i,i+1) and a(i,k+1). That is these two entries can be obtained by considering the product $Q^H_{(k,\ i)} Q^H_{(k,\ i-1)}\ R'_k\ Q_{(k,\ i-1)} Q_{(k,\ i)}$. However these two rotations affect only rows (i-1), i, and (i+1) and finding a(i,k+1) and b(i,k+1) reduces to the simple matrix multiplication of 3 x 3 matrices such that if we define

x(i,k) =x1, c(i, k) = c1, c(i-1, k) = c0, s(i, k)= s1 s(i-1, k) = s0,

a(i, k) = a1, a(i+1, k) = a2, b(i, k) = b1, b(i+1, k) = b2, a(i, k+1) = a3,

b(i, k+1) = b3, a(i,k+1) and b(i,k+1) can be calulated as

$$\begin{bmatrix} (.) & (.) & (.) \\ b3 & a3 & (.) \\ (.) & (.) & (.) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c1 & s1^* \\ 0 & -s1 & c1 \end{bmatrix} \begin{bmatrix} c0 & s0^* & 0 \\ -s0 & c0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x1 & (.) & (.) \\ b1 & a1 & b2^* \\ 0 & b2 & a2 \end{bmatrix} \begin{bmatrix} c0 & -s0^* & 0 \\ s0 & c0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c1 & -s1^* \\ 0 & s1 & c1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & c1 & s1^* \\ 0 & -s1 & c1 \end{bmatrix} \begin{bmatrix} c0 & s0^* & 0 \\ -s0 & c0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x1 & (.) & (.) \\ b1 & a1 & b2^* \\ 0 & b2 & a2 \end{bmatrix} \begin{bmatrix} c0 & -c1s0^* & -s0^* s1^* \\ s0 & c0c1 & -c0s1^* \\ 0 & s1 & c1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & c1 & s1^* \\ 0 & -s1 & c1 \end{bmatrix} \begin{bmatrix} c0x1+s0b1^* & (.) & (.) \\ 0 & -s0b1^* + c0a1 & c0b2^* \\ 0 & b2 & a2 \end{bmatrix} \begin{bmatrix} c0 & -c1s0^* & -s0^* s1^* \\ s0 & c0c1 & -c0s1^* \\ 0 & s1 & c1 \end{bmatrix}$$

By solving the above matrix for the value of b3 , we get

$$b3 = (0 , \ c1(-s0b1^* + c0a1) + s1^* b2, \quad c0c1b2^* + a2s1) \begin{bmatrix} c0 \\ s0 \\ 0 \end{bmatrix}$$

or

$$b(i, k+1) = s(i-1, k)[c(i, k)[c(i-1, k)a(i, k)-s(i-1, k)b^*(i, k)] + s^*(i,k)b(i+1, k) \qquad (2.24)$$

Let

$$w = c(i, k)[-s(i-1, k)\, b^*(i, k) + c(i-1, k)\, a(i, k)] + s(i, k)\, b(i+1, k)$$

Substituting w in (2.24), we get

$$b(i, k+1) = s(i-1, k) \cdot w$$

Similarly for a3, we have

$$a3 = (0 , \ c1(-s0b1^* + c0a1) + s1^* b2^*, \quad c0c1b2^* + a2s1^*) \begin{bmatrix} -c1c0^* \\ c0c1 \\ s1 \end{bmatrix}$$

or

$$a(i, k+1) = c(i-1, k)c(i, k)\{c(i, k)[c(i-1, k)a(i, k)-s(i-1, k)b^*(i, k)]+s^*(i, k) \qquad (2.25)$$

$b(i+1, k)\} + s(i, k)[c(i-1, k)c(i, k)b^{*}(i+1, k) + s^{*}(i, k)a(i+1, k)$

Let

$v = c(i-1, k)\, c(i, k)\, b^{*}(i+1, k) + s^{*}(i, k)\, a(i+1, k)$

Substituting w and v in equation (2.25) yields

$a(i, k+1) = c(i-1, k)\, c(i, k)\, .\, w + s(i, k)\, .\, v$

Similarly the values of s(i, k) and c(i, k) can be calculated using the following general relation.

$c(i, k) = x(i+1, k)/r$

$s(i, k) = b(i+1, k)/r$

where x(i+1, k) is the updated a(i, k) after $(i-1)^{th}$ rotation and is given by

$x(i+1, k) = -s(i-1, k)b(i, k) + c(i-1, k)a(i, k)$

and

$r = \mathrm{sqrt}(|b(i, k)|^{2} + x(i+1, k)^{2})$

Thus, if we denote the diagonal and subdiagonal elements entries of the matrix $T=T_1$ as a(i,0) and b(i,0) respectively, and the entries of the matrix U as u(i,j,0), a psedocode to update the Hermetian tridiagonal matrix and the matrix of eigenvectors is given as follows:

x(1, k)=0; b(1, k)=0; a(0, k)=0; b(m+1, k)=0; c(0, k)=1; s(0, k) = 0;

c(m+1, k)=1; s(m+1, k)=0;

k = 0

repeat

for i=1,m

$x(i+1, k) = -\, s(i-1, k)\, .\, b^{*}(i, k) + c(i-1, k)\, .\, a(i, k)$

$r = \mathrm{sqrt}\{\, |b(i, k)|^{2} + x(i+1, k)^{2}\, \}$

```
            if r > 0
                c(i, k) = x(i+1, k)/r
                s((i, k) = b(i+1, k)/r
            else
                c(i, k) = 1
                s(i, k) = 0
            end if
            w=c(i, k) . x(i+1, k)+s* (i, k) . b(i+1, k)
            v=c(i-1, k) . c(i, k) . b* (i+1, k) + s* (i, k) . a(i+1, k)
            b(i, k+1)=s(i-1, k) . w
            a(i, k+1)=c(i-1, k) . c(i, k) . w + s(i, k) . v
            for j=1,m
                u(i, j, k+1) = c(i, k) . u(i, j, k)+s* (i, k) . u(i+1, j, k)
                u(i+1, j, k+1) = -s(i, k) . u(i, j, k)+c(i, k) . u(i+1, j, k)
            end for
        end for
    k = k +1
    until sum of squares of b = 0
```

2.2 NON SYMMETRICAL SINGULAR VALUE DECOMPOSITION

Although, the generalized eigendecomposition is well defined for square matrices and has proven to be applicable to a variety of cases, it has a principal drawback of accuracy when applied to the data covariance matrix. The computation of the data covariance matrix involves implicit squaring of the data which may deteriorate numerical stability and accuracy of the eigenvectors and eigenvalues because of the ill-conditioned character of the matrix [11] . Under this condition, a

more fruitful approach to mitigate this problem to some extent is to operate directly on the observed data using the singular value decomposition (SVD).

Let $\mathbf{X} \in C^{m \times N}$ denote the data covariance matrix . The objective is to obtain a set of vectors spanning its columns space, that best approximate $\mathbf{X}$ in the least-square sense. Assume without loss of generality that N>m, then the singular value decomposition of

$\mathbf{X}/\sqrt{N}$ is

$$\mathbf{X}/\sqrt{N} = \mathbf{U}\,\Sigma\,\mathbf{V}^H \tag{2.26}$$

In the above equation $\mathbf{U}$ and $\mathbf{V}$ are mxm and NxN unitary matrices respectively, and $\Sigma = [\Sigma_1, 0]$, where Σ_1 is an mxm diagonal matrix with non-negative entries and 0 denotes an mx(N-m) matrix whose entries are zeros. It can be easily shown that the eigendecomposition and the SVD yield the same subspace estimate. In fact the data covariance matrix may be expressed as

$$\mathbf{R}_{xx} = \frac{\mathbf{X}}{\sqrt{N}}\frac{\mathbf{X}^H}{\sqrt{N}} = \frac{\mathbf{X}\mathbf{X}^H}{N} = \mathbf{U}\,\Sigma\,\mathbf{V}^H\,\mathbf{V}\Sigma^T\mathbf{Y}^H \tag{2.27}$$

$$= \mathbf{U}\,\Sigma\,\Sigma^T\mathbf{U}^H = \mathbf{U}\begin{bmatrix}\Sigma_1 & 0\end{bmatrix}\begin{bmatrix}\Sigma_1 \\ 0\end{bmatrix}\mathbf{U}^H = \mathbf{U}\,\Sigma_1^2\,\mathbf{U}^H \tag{2.28}$$

Thus the left singular vectors of $\mathbf{X}$ are the eigenvectors of $\mathbf{R}_{xx}$, and the eigenvalues of $\mathbf{R}_{xx}$ are the diagonal elements of Σ_1 . To obtain $\mathbf{U}$ and Σ_1, a first step would be to reduce $\mathbf{X}$ to square lower bidiagonal matrix $\mathbf{Y}$. This can be done by performing a preprocessing using Householder's transformations . The transformations from the left and right are determined to eliminate a whole column and row below and

above the subdiagonals without disturbing any previously zeroed rows or columns as in the case of the tridiagonal matrix. The basic iterative sequence of operations can be shown in the following example, where $\mathbf{X}$ is assumed to be a 4 x 6 matrix

$$\mathbf{X} = \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ x & x & x & x & x & x \\ x & x & x & x & x & x \end{bmatrix}$$

$$\Longrightarrow \quad \mathbf{X}\,\mathbf{N}_{(2,1)} = \begin{bmatrix} x & 0 & 0 & 0 & 0 & 0 \\ x & x & x & x & x & x \\ x & x & x & x & x & x \\ x & x & x & x & x & x \end{bmatrix}$$

$$\Longrightarrow \quad \mathbf{N}_{(1,1)}\,\mathbf{X}\,\mathbf{N}_{(2,1)} = \begin{bmatrix} x & 0 & 0 & 0 & 0 & 0 \\ x & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & x & x & x & x & x \end{bmatrix}$$

$$\Longrightarrow \mathbf{N}_{(1,1)}\,\mathbf{X}\,\mathbf{N}_{(2,1)}\mathbf{N}_{(2,2)} = \begin{bmatrix} x & 0 & 0 & 0 & 0 & 0 \\ x & x & 0 & 0 & 0 & 0 \\ 0 & x & x & x & x & x \\ 0 & x & x & x & x & x \end{bmatrix}$$

$$\Longrightarrow \mathbf{N}_{(1,2)}\mathbf{N}_{(1,1)}\,\mathbf{X}\,\mathbf{N}_{(2,1)}\mathbf{N}_{(2,2)} = \begin{bmatrix} x & 0 & 0 & 0 & 0 & 0 \\ x & x & 0 & 0 & 0 & 0 \\ 0 & x & x & x & x & x \\ 0 & 0 & x & x & x & x \end{bmatrix}$$

$$\Longrightarrow \mathbf{N}_{(1,2)}\mathbf{N}_{(1,1)}\,\mathbf{X}\,\mathbf{N}_{(2,1)}\mathbf{N}_{(2,2)}\,\mathbf{N}_{(2,3)} = \begin{bmatrix} x & 0 & 0 & 0 & 0 & 0 \\ x & x & 0 & 0 & 0 & 0 \\ 0 & x & x & 0 & 0 & 0 \\ 0 & 0 & x & x & x & x \end{bmatrix}$$

$$==> \mathbf{N}_{(1,2)}\mathbf{N}_{(1,1)}\ \mathbf{X}\ \mathbf{N}_{(2,1)}\mathbf{N}_{(2,2)}\ \mathbf{N}_{(2,3)}\ \mathbf{N}_{(2,1)} = \begin{bmatrix} x & 0 & 0 & 0 & 0 & 0 \\ x & x & 0 & 0 & 0 & 0 \\ 0 & x & x & 0 & 0 & 0 \\ 0 & 0 & x & x & 0 & 0 \end{bmatrix}$$

Let $\mathbf{N}_1^H = \mathbf{N}_{(1,2)}\mathbf{N}_{(1,1)}$ and $\mathbf{N}_2^H = \mathbf{N}_{(2,1)}\mathbf{N}_{(2,2)}\ \mathbf{N}_{(2,3)}\ \mathbf{N}_{(2,4)}$

then $\mathbf{N}_1^H\ \mathbf{X}\ \mathbf{N}_2^H = [\ \mathbf{Y},\ 0\ ]$ (2.29)

Where $\mathbf{Y}$ is a lower bidiagonal matrix of order m. Then two matrices $\mathbf{A}$ and $\mathbf{B}$ can be detremined, using Given's rotations, to eliminate the unwanted nozeros elements down the diagonal, such that

$$\Sigma_1 = \mathbf{A}^H \mathbf{Y}\ \mathbf{B}$$

or

$$\mathbf{Y} = \mathbf{A}\,\Sigma_1\,\mathbf{B}^H \tag{2.30}$$

The substitution of (2.30) into (2.29) yields

$$\mathbf{N}_1^H\ \mathbf{X}\ \mathbf{N}_2^H = [\ \mathbf{A}\Sigma_1\mathbf{B}^H,\ 0\ ] \tag{2.31}$$

or

$$\mathbf{X} = \mathbf{N}_1 [\ \mathbf{A}\Sigma_1\mathbf{B}^H,\ 0\ ]\ \mathbf{N}_2 \tag{2.32}$$

but

$$[\ \mathbf{A}\Sigma_1\mathbf{B}^H,\ 0\ ] = [\ \mathbf{A}\Sigma_1,\ 0\ ] \begin{bmatrix} \mathbf{B}^H & 0 \\ 0 & 0 \end{bmatrix} \tag{2.33}$$

thus

$$\mathbf{X} = \mathbf{N}_1 \; [\; \mathbf{A}\,\boldsymbol{\Sigma}_1 \,,\, \mathbf{0} \;] \begin{bmatrix} \mathbf{B}^H & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{N}_2 \tag{2.34}$$

If we let

$$\mathbf{N}_1 \; \mathbf{A} = \mathbf{U} \text{ and } \begin{bmatrix} \mathbf{B}^H & 0 \\ 0 & 0 \end{bmatrix} \mathbf{N}_2 = \mathbf{V}^H \tag{2.35}$$

we find the formula given by (2.26). For the computation of the eigenvector matrix , one needs only to store the matrix $\mathbf{N}_1$ which together with A give the matrix U.

Although, the SVD assures better accuracy of the eigenvectors and eigenvalues, but it requires a large memory to store the data matrix, and extensive computations on the matrix $\mathbf{X}$ to reduce it to a bidiagonal one.

Chapter 3

PARALLEL ARCHITECTURE FOR MUSIC ALGORITHM

3.1 INTRODUCTION

With the advances in the area of VLSI it is now possible to design special purpose hardware for the implementation of various real time algorithms. The customized hardware has two main advantages as listed below.

1) The given algorithm is executed at a high speed.
2) Cost and size of the hardware will be lower than the cost of a general purpose computer.

These advantages have led many researchers to probe into the possibility of designing special purpose hardware. The development of special purpose hardware will need to exploit pipeline, parallel and distributed processing approaches to achieve high throughput rates. Hence in this section we present the first step in the development of a dedicated chip set to support MUSIC and ESPRIT algorithm which has been explained in previous sections.

There are many architectures such as systolic array, SIMD Cordic Processors and MIMD which can be used for parallel implementation. An appropriate structure which can exploit maximum parallelization to reduce the computation time will be selected for real time implementation for the particular application.

3.2 LITERATURE SEARCH

Various papers pertaining to parallelization of Householders and QR algorithms were reviewed. C.F.T. Tang et al [12] and K.J.R. Liu [13] proposed architecture for complex Householder transformations for triangularization of the matrix. In their architecture they used single column with the number of processors equal to the number of columns of the matrix. Each processor performs operation on each column. After each iteration the values of each column are fed back to the same processors. But their architecture is proposed to perform triangularization of the given matrix whereas we are interested in tridiagonalization of the covariance matrix.

QR method for the tridiagonal matrix is implemented by W. Phillips [15]. In his architecture, rectangular systolic array is used in which each processor performs single iteration. When the first iteration is performed on the m th row by the k th processor, the second iteration is performed on the (m-1) th row by the (k-1) th processor and so on. But the disadvantage in this approach is that the number of processors is dependent on the the number of iterations, i.e., if 5 iterations are required then 5 processors are required. But the exact number of iterations is not known which leads to the uncertainity of the required number of processors.

K.J.R.Liu [16] has proposed another kind of approach in which a systolic array arranged in a matrix form is used. The number of processors is equal to the number of elements of the matrix. During first step, the matrix Q is found. Then new values of matrix A are then calculated using Q. Convergence for all the elements of the matrix other than the diagonal elements is checked. If all

the elements of the matrix other than the diagonal elements are not equal to zero then the same systolic array is used for the next iteration. These iterative computations are used until all the elements of the matrix except the diagonal elements converge to zero. The obvious advantage is that the same set of processors can be used for all the iterations. But the drawback is that this architecture is proposed for the evaluation of eigenvalues on the dense matrix. In the next sub-section systolic architecture for the previously developed parallel algorithms for the computations of covariance matrices, householders transformation and QR method is presented.

3.3 HARDWARE BLOCK DIAGRAM OF MUSIC AND ESPRIT ALGORITHMS

The hardware block diagram for the MUSIC Algorithm is shown in Figure 3.1. As seen in this figure, the data collected from the sensors is utilized to form the covariance matrix. The eigendecomposition is performed using Householders transformation and QR method. The Eigenvalues are used to find the number of sources and finally using the eigenvectors in Power method we find the angle of arrival. The Hardware block diagram for ESPRIT algorithm is shown in Figure 3.2. It is similar to the MUSIC algorithm but instead of power method of MUSIC some more computations are performed to evaluate the angle of arrival in case of ESPRIT.

3.2 DATA COVARIANCE MATRIX FORMATION

Once the data has been collected by the sensors the data covariance matrix can be computed using

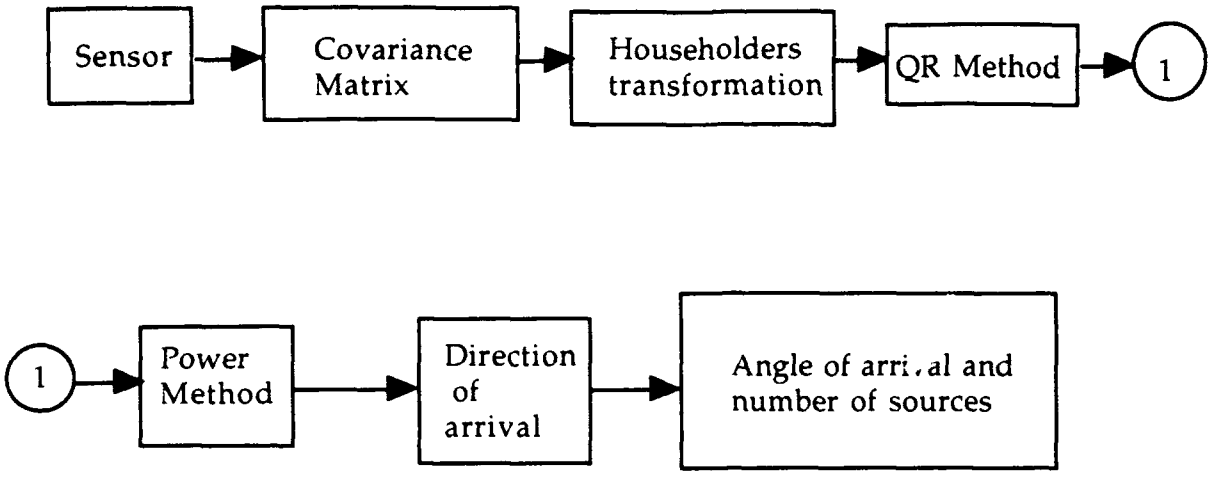


Figure 3.1: Hardware block diagram for MUSIC algorithm

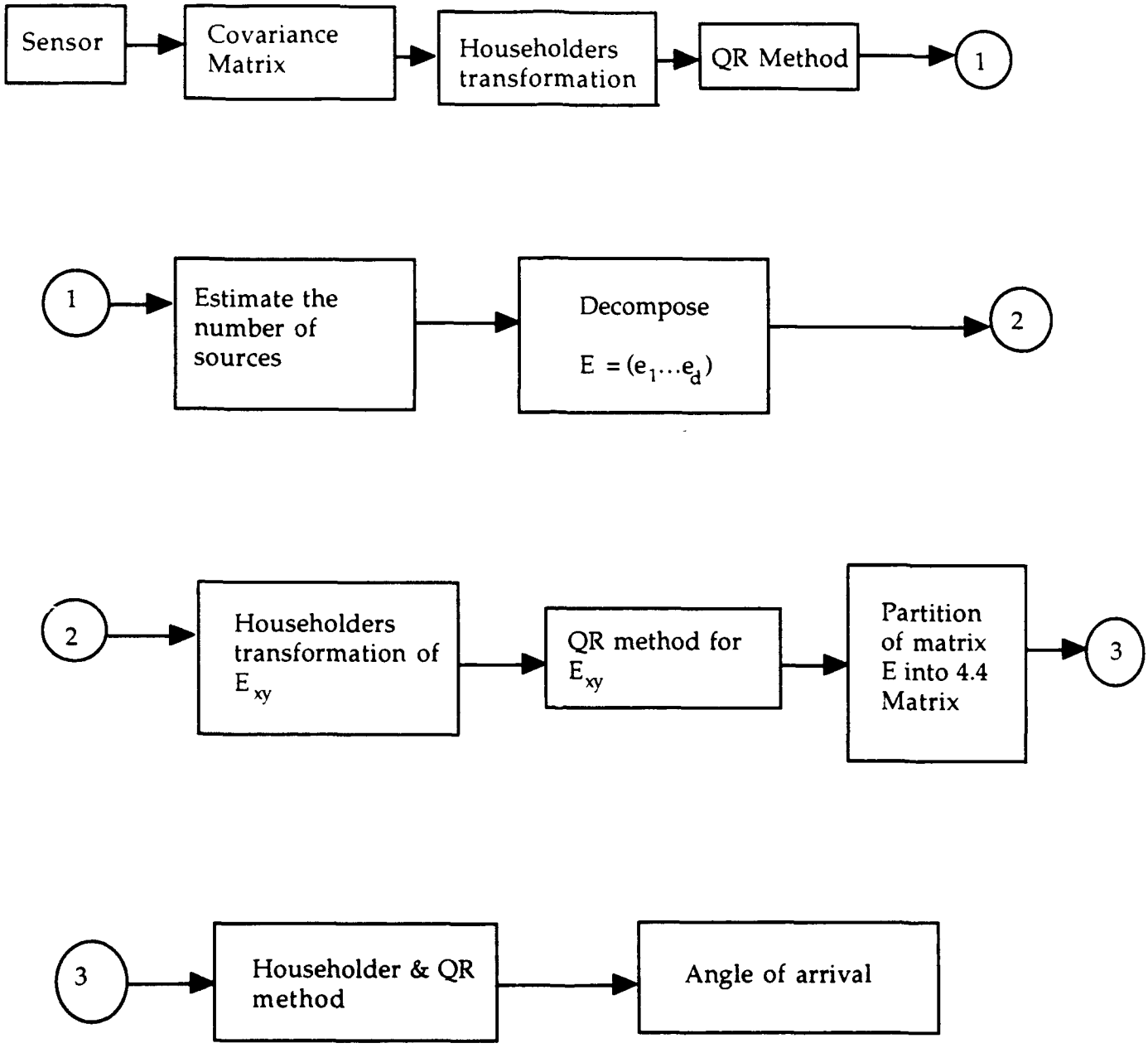


Figure 3.2: Hardware block diagram of ESPRIT algorithm.

$$\mathbf{R}_{xx} = \frac{\sum_{p=1}^{n} x(p).x(p)^*}{n} \qquad (3.1)$$

Where $\mathbf{R}_{xx}$ = Covariance of data matrix

$x(p)$ = Vector of data output from every sensor at p^{th} sample given by $(x_1, x_2, \ldots\ldots x_8)^T$

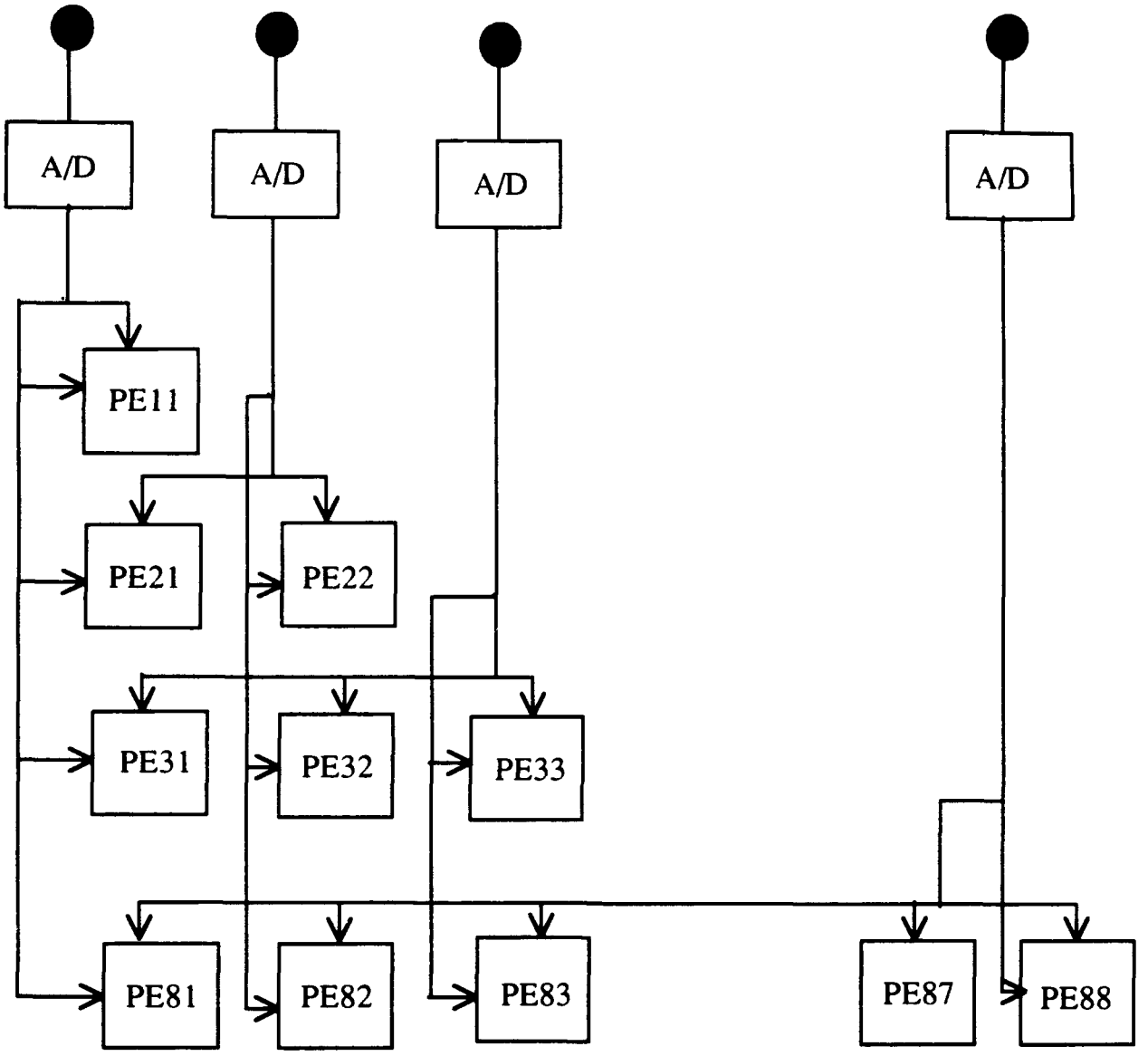
n = Number of samples

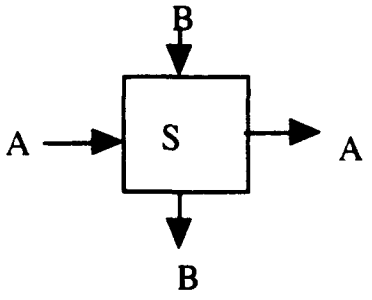
The sampled data obtained from the sensors is used to obtain the data covariance matrix given by equation (3.1). For instance, the element (i,j) of $\mathbf{R}_{xx}$ denoted by $\mathbf{R}_{ij}$ is computed as:

$$\mathbf{R}_{ij} = \frac{\sum_{p=1}^{n} x_i(p) \cdot x_j^*(p)}{n}$$

Since the covariance matrix is Hermetian, the computation of lower triangular matrix of covariance matrix is sufficient to get complete information of the full matrix.

The parallel computation of the data covariance matrix is performed using systolic architecture. As stated earlier the covariance matrix is Hermetian and computation of lower triangular elements of the covariance matrix is sufficient to get the information for the entire matrix. Since there are 36 elements in the lower triangular 8×8 matrix, systolic architecture will have 36 processors. Here a triangular arrangement of the systolic array with global routing is considered as shown in Figure 3.3. Each processor is numbered as P_{mn} where m is the row number and n is the column number. The sampled data from the i^{th} sensor is sent






```
S = 0
do i=1,p
S = S + A . B
end do
S = S ÷ p
```

Figure 3.3: Architecture for Computation of Covariance Matrix

to the i^{th} row and the i^{th} column simultaneously. For example the sampled data from the 3^{rd} sensor is sent to all the processors in the third row and the third column. Each processor performs multiplication and addition of two sampled data in parallel in all the processors for every clock cycle . Since there are 36 processors, 36 multiplications and 36 additions are performed simultaneously. Each processor has a memory to store the product of multiplication which is added to the product obtained during the next data cycle. Once the operations of multiplication and addition for all the sampled data in all the processors is performed, the stored data in each processor is then divided by the number of samples in all the processors in parallel. The resulting output are used to form the data covariance matrix R_{xx}.

3.4 HARDWARE FOR HOUSEHOLDER TRANSFORMATIONS

As shown in Chapter 2, the determination of all the **d** and the new elements of the columns of the matrix can be computed in parallel. A modified architecture is proposed for the computation of the tridiagonal matrix. Thus this algorithm can be mapped on a hardware architecture with the number of processors equal to m+1, where m is the order of the matrix. Arrangement for 8×8 matrix is as shown in Figure 3.4. The columns of the matrix are sent to each processor in a pipelined fashion in reverse order such that the last element of each column becomes the first element. The Processor PE1 is used to find the **w** and c required by other processors to find the **d**. Processors PE2, PE3... PE8 are used to find **d** using the value of **w** and c found in the first processor. At the same time the first processor is used for the evaluation of β. The first element of the first column and β are the output of the first iteration which are used as input for evaluation of eigenvalues using QR method. All the d_s are evaluated in parallel and are sent to

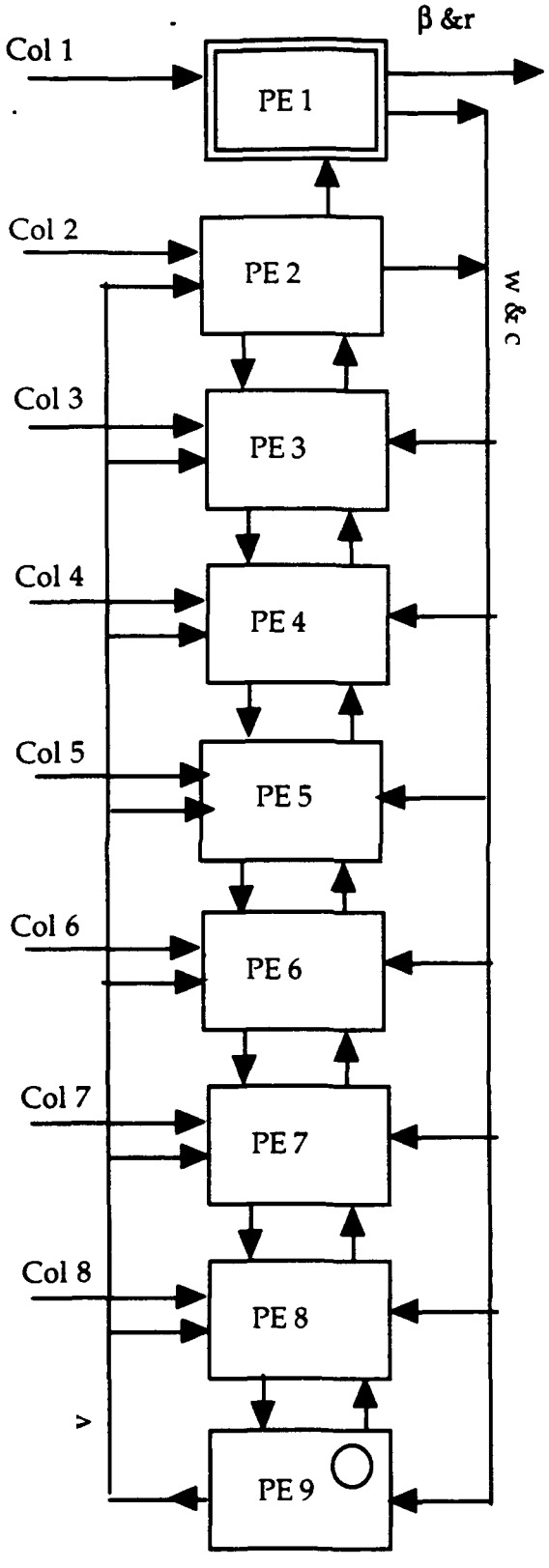


Figure 3.4: Architecture For Householders Method

the processor PE9. The processor PE9 is exclusively used for the determination of **v** using **d** and **w**. The **v** are then routed back to all the processors. The processors PE2, PE3... PE8 use **w**, **d** and **v** to find the new values of the elements of the columns in parallel. The counter is used to set the number of iterations to m-2. For m-2 times, the intermediate results are used in feedback loop and the same set of processors are used repeatedly. The feedback loop has a FIFO memory to temporally store all the elements of the column until operations on previous iteration are completed. For the first iteration, operations on 8×8 matrix are performed hence all the processors are utilized. For the second iteration, operation on 7×7 matrix are performed. Now the first column of the matrix is already computed; therefore new elements of the second column from PE2 are fed back to PE. Thus for the second iteration, PE2 does not have any column to work on and is thus disabled. All other processors perform same operation as in the first iteration, but the elements of each column are reduced by one element. Thus for every new iteration the columns and the elements of the columns keeps on reducing.

3.5 PARALLEL ALGORITHM FOR THE TRIDIAGONAL QR ALGORITHM

The given factorization, if applied to a full m x m data covariance matrix $\mathbf{R_{xx}}$, will result in the operational cost of every factorization being O (m^3) [42]. This follows from the QR algorithm where by setting $\mathbf{A_1} = \mathbf{R_{xx}}$, the first phase consists of calculating an upper triangular matrix $\mathbf{R_k}$ and a unitary matrix $\mathbf{Q_k}$ such that $\mathbf{A_k} = \mathbf{Q_k}$ $\mathbf{R_k}$ and during the second phase, the product $\mathbf{R_k}$ $\mathbf{Q_k}$ is performed to obtain A_{k+1} k=1,..., n-1, where n denotes the number of iterations. For this reason, it is generally not feasible to carry out the QR transformations on $\mathbf{R_{xx}}$ Instead, if $\mathbf{R_{xx}}$ is first reduced to a tridiagonal matrix **T** using Householder's transformations, the subsequent transformations in chapter 2 will always give a tridiagonal matrix, and

thus, only (m-1) subdiagonal elements have to be eliminated to obtain the matrix $\mathbf{R}_k$ during every factorization. Thus, the cost of the eigendecomposition falls substantially from $O(m^3)$ to O(m) [42]. Furthermore, Phillips and Roberston [15] proposed a sequential algorithm for updating the entries of the tridiagonal matrix without calculating, in the first step, the matrix $\mathbf{R}_k$. Although this algorithm reduces the processing time to some extent by avoiding the computation of R_k by Q_k at every iteration, and also the storage of the matrix $\mathbf{R}_k$, but still every iteration in the algorithm requires m steps. For n iterations, mxn steps are required to perform the eigendecomposition of the tridiagonal matrix. For the case of a matrix of order 8 and for 11 iterations, various steps are shown in Figure 3.5, where (i,j) denotes the pairs of a(i,.) and b(i, .) at the j^{th} step.

At every step, one pair of a's and b's are computed. For example, in step 5 a(5,1) and b(5,1) are computed. Every iteration requires eight steps . The last elements shown in Figure 3.5 are a(8,11) and b(8,11), and they are computed after 88 computation steps. In this section, an attempt is made to parallelize this sequential algorithm to reduce the number of steps from mxn to 2(m+n)-10 steps. A parallel/pipelined algorithm has been developed and is described in terms of a simple program consisting of odd and even steps. During an odd step, the odd terms (a(i,.), b(i,.)), i=1,3,...m-1, of the matrix T given by (2.22) are updated in parallel. Likewise, during an even step, the even terms a(i,.), b(i,.), i=2,4...,m, are updated in a similar fashion. A pseudocode of this algorithm is given in the following :

Step	Computation performed sequentially
1	(1,1)
2	(2,1)

3				(3,1)					
4					(4,1)				
5						(5,1)			
6							(6,1)		
7								(7,1)	
8									(8,1)
9		(1,2)							
10			(2,2)						
11				(3,2)					
12					(4,2)				
13						(5,2)			
14							(6,2)		
15								(7,2)	
16									(8,2)
~~	~~	~~	~~	~~	~~	~~	~~	~~	~~
81		(1,11)							
82			(2,11)						
83				(3,11)					
84					(4,11)				
85						(5,11)			
86							(6,11)		
87								(7,11)	
88									(8,11)

Figure 3.5 . Example of the sequential algorithm for updating the entries a's and b's of a tridiagonal matrix (matrix order = 8,number of iteration = 11)

```
x(1,i)=0;b(1,i)=0;a(0,i)=0; b(m+1,i)=0;c(0,i)=1;s(0,i)=0;
c(m+1,i)=1; s(m+1,i)=0;
n = number of iterations
for k= 1,n+(m-2)/2

 Odd Steps

        for j=1,m-1,2
                i=k-(j+1)/2
Update the a's and b's
                if  (i<0)
                        b(j,i+1)=b(j,0)
```

```
                a(j,i+1)=a(j,0)
            else if ( i  > n-1)
                b(j,i+1)=b(j,n)
                a(j,i+1)=a(j,n)
            else
                x(j+1,i)=-s(j-1,i).b*(j,i)+c(j-1,i).a(j,i)
                if  ( x(j+1,i) = 0)
                    c(j,i)=1
                    s(j,i)=0
                else
                    r=sqrt( |b(j+1,i)|^2  +x(j+1,i)^2 )

                    c(j,i)=x(j+1,i)/r
                    s(j,i)=b(j+1,i)/r
                endif
                w=c(j,i) x(j+1,i)+s*(j,i) b(j+1,i)
                v=c(j-1,i) c(j,i) b*(j+1,i) + s*(j,i) a(j+1,i)
                b(j,i+1)=s(j-1,i) w
                a(j,i+1)=c(j-1,i) c(j,i) w + s(j,i) v
```

Computations of eigenvectors

```
                for l=1,m
                    u(j,l,i+1)=c(j,i)u(j,l,i).+s*(j,i)  u(j+1,l,i)
                    u(j+1,l,i+1) = -s(j,i) u(j,l,i)+c(j,i) u(j+1,l,i)
                end for
            endif
        end for
```

Even steps

```
        for j=2,m,2
            i=k - j/2
```

Computation of the a's and b's

```
            if  (i<0)
                b(j,i+1)=b(j,0)
                a(j,i+1)=a(j,0)
            else if ( i  > n-1)
                b(j,i+1)=b(j,n)
                a(j,i+1)=a(j,n)
            else
                x(j+1,i)=-s(j-1,i).b*(j,i)+c(j-1,i).a(j,i)
```

```
                    if ( x(j+1,i) = 0)
                        c(j,i)=1
                        s(j,i)=0
                    else
                        r=sqrt( |b(j+1,i)|^2 +x(j+1,i)^2 )

                        c(j,i)=x(j+1,i)/r
                        s(j,i)=b(j+1,i)/r
                    endif
                    w=c(j,i) x(j+1,i)+s*(j,i) b(j+1,i)
                    v=c(j-1,i) c(j,i) b*(j+1,i) + s*(j,i) a(j+1,i)
                    b(j,i+1)=s(j-1,i) w
                    a(j,i+1)=c(j-1,i) c(j,i) w + s(j,i) v
```

Computations of eingenvectors

```
                    for l=1,m
                        u(j,l,i+1)=c(j,i)u(j,l,i).+s*(j,i) u(j+1,l,i)
                        u(j+1,l,i+1) = -s(j,i) u(j,l,i)+c(j,i) u(j+1,l,i)
                    end for
                endif
            end for
        endfor
```

An example of this algorithm applied to a matrix of order 8, and for 11 iterations is shown in Figure 3.6, where (i ,j) is defined earlier.

Step	Compuations performed in parallel							
1	(1,1)							
2		(2,1)						
3	(1,2)		(3,1)					
4		(2,2)		(4,1)				
5	(1,3)		(3,2)		(5,1)			
6		(2,3)		(4,2)		(6,1)		
7	(1,4)		(3,3)		(5,2)		(7,1)	
8		(2,4)		(4,3)		(6,2)		(8,1)
9	(1,5)		(3,4)		(5,3)		(7,2)	
10		(2,5)		(4,4)		(6,3)		(8,2)
11	(1,6)		(3,5)		(5,4)		(7,3)	
12		(2,6)		(4,5)		(6,4)		(8,3)
13	(1,7)		(3,6)		(5,5)		(7,4)	
14		(2,7)		(4,6)		(6,5)		(8,4)
15	(1,8)		(3,7)		(5,6)		(7,5)	
16		(2,8)		(4,7)		(6,6)		(8,5)
17	(1,9)		(3,8)		(5,7)		(7,6)	
18		(2,9)		(4,8)		(6,7)		(8,6)
19	(1,10)		(3,9)		(5,8)		(7,7)	
20		(2,10		(4,9)		(6,8)		(8,7)
21	(1,11)		(3,10)		(5,9)		(7,8)	
22		(2,11)		(4,10)		(6,9)		(8,8)
23			(3,11)		(5,10)		(7,9)	
24				(4,11)		(6,10)		(8,9)
25					(5,11)		(7,10)	
26						(6,11)		(8,10)
27							(7,11)	
28								(8,11)

Figure 3.6 . Example of the parallel/pipelined algorithm for updating the entries a's and b's of a tridiagonal matrix (matrix order = 8,number of iteration = 11)

In the following, some of the computations performed by this parallel/pipelined algorithm are given below

Computations performed during step1

$$x(2,0)=-s(0,0).b^*(1,0)+c(0,0).a(1,0)$$

```
if ( x(2,0) = 0)
        c(1,0)=1
        s(1,0)=0
else
        r=sqrt(| |b(2,0)| |^2+x(2,0)^2)

        c(1,0)=x(2,0)/r
        s(1,0)=b(2,0)/r
endif
w=c(1,0).x(2,0)+s*(1,0) b(2,0i)
v=c(0,0) . c(1,0) b*(2,0) + s*(1,0) a(2,0)
b(1,1)=s(0,0) . w
a(1,1)=c(0,0) c(1,0) w + s(1,0) v
```

Computations performed during step 2

```
x(3,0)=-s(1,0).b*(2,0)+c(1,0).a(2,0)
if ( x(3,0) = 0)
        c(2,0)=1
        s(2,0)=0
else
        r=sqrt(| |b(3,0)| |^2+x(3,0)^2)

        c(2,0)=x(3,0)/r
        s(2,0)=b(3,0)/r
endif
w=c(2,0) x(3,0)+s*(2,0) b(3,0)
v=c(1,0),c(2,0) b*(3,0) + s*(2,0) a(3,0)
b(2,1)=s(1,0) w
a(2,1)=c(1,0) c(2,0) w + s(2,0) v
```

Computations performed during step 3

```
x(4,0)=-s(2,0).b*(3,0)+c(2,0).a(3,0)
If( x(4, 0) = 0)
        c(3,0)=1
        s(3,0)=0
else
        r=sqrt(| b(4,0)|^2+x(4,0)^2)

        c(3,0)=x(4,0)/r
        s(3,0)=b(4,0)/r
endif
w=c(3,0) x(4,0)+s*(3,0) b(4,0)
v=c(2,0) c(3,0) b*(4,0) + s*(3,0) a(4,0)
```

```
b(3,1)=s(2,0) . w
a(3,1)=c(2,0) c(3,0) w + s(3,0) v

x(2,1)=-s(0,1).b*(1,1)+c(0,1).a(1,1)
if ( x(2,1) = 0)
        c(1,1)=1
        s(1,1)=0
else
        r=sqrt( | b(3,0) | 2+x(3,0) 2)

        c(1,1)=x(2,1)/r
        s(1,1)=b(2,1)/r
endif
w=c(1,1).x(2,1)+s*(1,1) b(2,1)
v=c(0,1) c(1,1) b*(2,1) + s*(1,1) a(2,1)
b(1,2)=s(0,1) . w
a(1,2)=c(0,1) c(1,1) w + s(1,1) v
```

Computations performed during step 4

```
x(5,0)=-s(3,0).b*(4,0)+c(3,0).a(4,0)
if ( x(4, 0) = 0)
        c(4,0)=1
        s(4,0)=0
else
        r=sqrt( | b(5,0) | 2+x(5,0) 2)

        c(4,0)=x(5,0)/r
        s(4,0)=b(5, 0)/r
endif
w=c(4,0).x(5,0)+s*(4,0).b(5,0)
v=c(3,0).c(4,0) b*(5,0) + s*(4,0) a(5,0)
b(4,1)=s(3,0) w
a(4,1)=c(3,0) c(4,0) w + s(4,0) v

x(3,1)=-s(1,1).b*(2,1)+c(1,1).a(2,1)
if ( x(3,1) = 0)
        c(2,1)=1
        s(2,1)=0
 else
        r=sqrt( | b(3,1) | 2+x(3,1) 2)

        c(2,1)=x(3,1)/r
        s(2,1)=b(3,1)/r
endif
```

w=c(2,1) x(3,1)+s*(2,1) b(3,1)
v=c(1,1) .c(2,1) b*(3,1) + s*(2,1) a(3,1)
b(2,2)=s(1,1) w
a(2,2)=c(1,1) c(2,1) w + s(2,1) v

The algorithm updates the pairs of a's and b's in the following manner

Step 1 and 2	one pair
Step 3 and 4	two pairs
Step 5 and 6	three pairs
Step 7 to step 22	four pairs
Step 23 and 24	Three pairs
Step 25 and 26	two pairs
Step 27 and 28	one pair

This algorithm is also suitable for VLSI implementation, using an array of $m/2$ processors Pr_1, $Pr_{2,}$..., $Pr_{m/2}$ and $(m+2)$ cells cl_0, cl_1,..., c_{m+1} constituting a local memory, as shown in Figure 3.7. Each processor in the array performs certain computations such as floating point operations and square roots.

If the pairs (a(i,.), b(i,.)), (c(i,.), s(i,.)) , i=0,2,...,m+1 , are stored respectively in cl_0 cl_1,..., cl_{m+1}, then during an odd step , each processor Pr_i, respectively

1) accepts

   a) c(2i-2, k), s(2i-2, k) from cell cl_{2i-2}

   b) a(2i-1,k), b(2i-1,k) from cell cl_{2i-1}

   c) a(2i,k), b(2i,k) from cell cl_{2i}

2) computes x(2i,k), c(2i-1,k), and s(2i-1,k)

3) updates a(2i-1,k) and b(2i-1,k) to become a(2i-1, k+1) and b(2i-1, k+1) respectively

4) stores c(2i-1,k), s(2i-1,k), a(2i-1,k+1), and b(2i-1, k+1) in cell cl_{2i-1}

and during an even step , each processor Pr_i , respectively

1) accepts

a) c(2i-1, k), s(2i-1, k) from cell cl_{2i-1}

b) a(2i,k), b(2i,k) from cell cl_{2i}

c) a(2i+1,k), b(2i+1,k) from ceil cl_{2i+1}

2) computes x(2i+1,k), c(2i,k), and s(2i,k)

3) updates a(2i,k) and b(2i,k) to become a(2i, k+1) and b(2i, k+1) respectively

4) stores c(2i,k), s(i,k), a(2i,k+1), and b(2i, k+1) in cell cl_{2i}

In the first step of the algorithm, the pairs (c(0,0)=1, s(0,0)=0), (a(1,0), b (1,0)), and (a(2,0) , b(2,0)) stored in cl_0, cl_1 and cl_2 respectively, are entered in the first processor Pr_1. These values are used to form x(2,0), and to compute c(1,0) and s(1,0) according to the algorithm. These two values (c(1,0),s(1,0)) are stored in cl_1 to be used during the next step. The computed values of c(1,0) and s(1,0) are used along with c(0,0), s(0,0), x(2,0), a(2,0) and b(2,0) to update a(1,0) and b(1,0) to become a(1,1) and b(1,1)

At the second step, the first processor accepts c(1,0), s(1,0), a(2,0) , b(2, 0), a(3,0) and b(3, 0) to update in a similar fashion a(2,0), and b(2,0) to a(2,1), and b(2,1).
At the third step, while the second processor is updating a(3,0) and b(3,0) to become a(3,1) and b(3,1) respectively, the first processor is used to update a(1,1) and b(1,1) to become a(1,2) and b(1,2) respectively.

At the fourth step, the second and first processors update in parallel the two pairs (a(4,0), b(4,0)) and (a(2,1), b(2,1)) to (a(4,1), b(4,1)) and (a(2,2), b(2,2)) respectively. The algorithm proceeds in this fashion so that the array is updating the entries of

the tridiagonal matrix in a pipeline fashion in the iteration as shown in Figure 3.8 for m=8 and for 11 iterations.

It can be noticed that after 28 steps, the updated entries of the matrix T are obtained. This result can be generalized to any matrix of order m, and for any number of iterations n, where it is not difficult to show that 2(n+m)-10 steps are needed to achieve the desired result. Although, the preceding method would serve to correctly obtain the updated entries after a fixed number of iterations, it can be extended to accomplish the same task for an unlimited number of iterations, until the convergence is satisfied.

The previous array , shown in Figure 3.9, can be extended to include another m/2 processors P_1 ,P_2 , ...,$P_{m/2}$, as shown in Figure 3.9, to update the matrix of the eigenvectors. Given the matrix $\mathbf{U} = \mathbf{N}^H$, obtained from Householder's transformations, and the matrix $\mathbf{Q} = \mathbf{Q}_1 \mathbf{Q}_2 \ldots \mathbf{Q}_n$, where n is the number of iterations. The product of $\mathbf{Q}^H$ by **U**, to obtain the matrix of eigenvectors of the original problem, may be computed also in 2n+m-2 steps. If each column of the matrix $U = N^H$ is stored in an array of m elements consisting of a FIFO as depicted in Figure 3.8 , then during an odd step, the values stored at the top of the independent pairs of arrays (1,2), (3,4),..., (m-1, m) are transfered in parallel to the processors P_1 ,P_2 , ...,$P_{m/2}$ respectively. The rotation parameters generated during this particular step are also sent to the corresponding processors. That is, the rotation parameters generated by Pr_1 are sent to P_1 , and the rotation parameters generated by Pr_2 are sent to P_2, and so on. Once the top elements are updated, they are transferred to the bottom of the corresponding arrays The procedure continues until all the elements stored in the array are updated This is depicted in Figure 3.6(a). Similarly, during

an even, updating the entries of the independent column pairs (2,3), (4,5),..., (m-2, m-1) is shown in Figure 3.6 (b)

3.6 HARDWARE IMPLEMENTATION OF THE POWER METHOD

Once the eigenvectors have been computed, the values of the eigenvectors are utilized to calculate the power

$$P_m(\theta) = \frac{1}{a^*(\theta)\, E_n E^*_n\, a(\theta)} \qquad (3.2)$$

As seen from this equation the evaluation of $a^*(\theta)\, E_n E^*_n\, a(\theta)$ requires squaring of the product of rowvector of $\mathbf{a}(\theta)$ and the eigenvector matrix $\mathbf{E}_n$. Hence the product of $\mathbf{a}(\theta)$ and $\mathbf{E}_n$ is evaluated and then the product obtained is squared and accumulated for all the values in the array manifold. The hardware design is shown in Figure 3.10. It consists of a set of 8 processors. Each processor finds the product of $\mathbf{a}(\theta)$ and columns $\mathbf{E}_n$ in parallel. The product obtained is then summed using adders. The evaluation $E^*_n\, a(\theta)$ is similar to the evaluation of $a^*(\theta)\, E_n$ and hence the product of $a^*(\theta)\, E_n$ is squared and added. The angle of arrival is thus calculated for different values of the array manifold. This computed value $Q_m(\theta)$ is inverse of $P_m(\theta)$ is different for different angle. The angle for which $Q_m(\theta)$ is minimum is the angle of arrival.

3.7 CONCLUSIONS

Flow diagram for the entire MUSIC algorithm is shown in Figure 3.11. Different stages in the MUSIC can be substituted by different hardware architecture as explained above. Hence the entire pipelined stage of MUSIC algorithm consist of

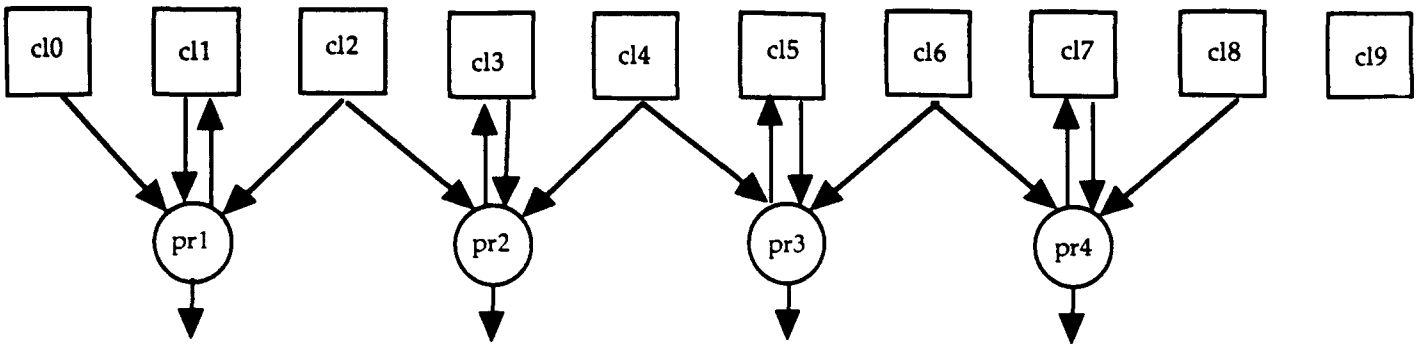
1) Data covariance matrix stage

2) Householders stage

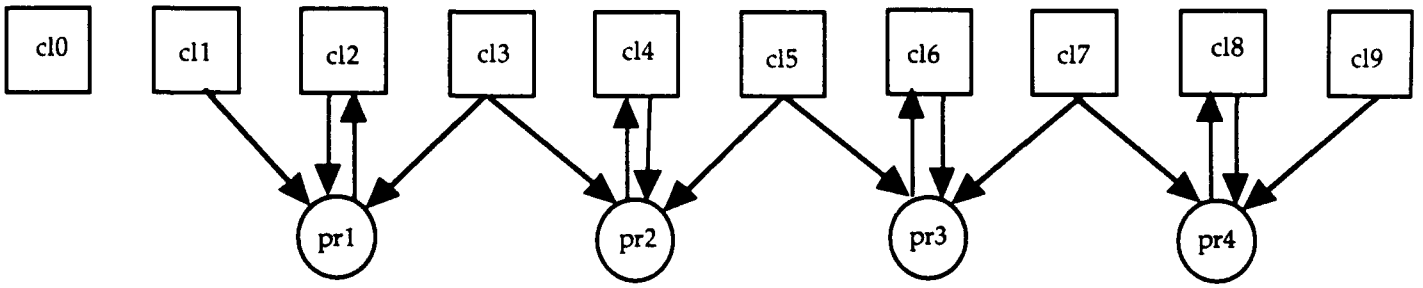
3) QR stage

4) Power method stage

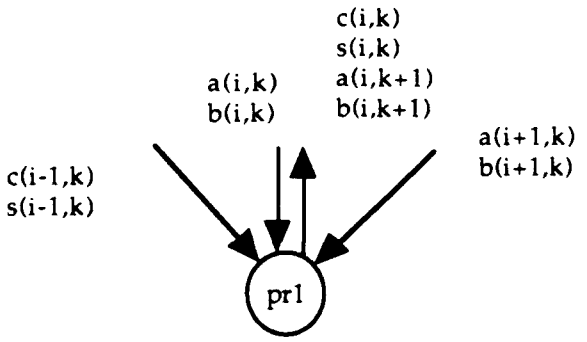
Various stages along with buffers ate explained in next chapter.



(a)



(b)




```
x(i+1,k)=-s(i-1,k)  b*(i,k)+c(i-1,k)a(i,k)
if (x(i+1,k) = 0)
    c(i,k)=1.
    s(i,k)=0.
else
     r = sqrt(|b(i+1,k)|.|b(i+1,k)|   + x(i+1,k). x(i+1,k))
     c(i,k)= x(i+1,k)/r
     s(i,k)= b(i+1,k)/r
end if
      w= c(i,k) x( i+1,k) +s*(i,k)b(i+1,k)
      v= c(i-1,k)c(i,k)b*(i+1,k) + s*(i,k)a(i+1,k)
      b(i,k+1)=s(i-1,k) w
```

Figure. 3 . Updating the eigenvalues, (a) odd steps, (b) even stepsa(i,k+1)=c(i-1,k)c(i,k)w+s(i,k)v

Figure. 3.6 . Updating the eigenvalues, (a) odd steps, (b) even steps

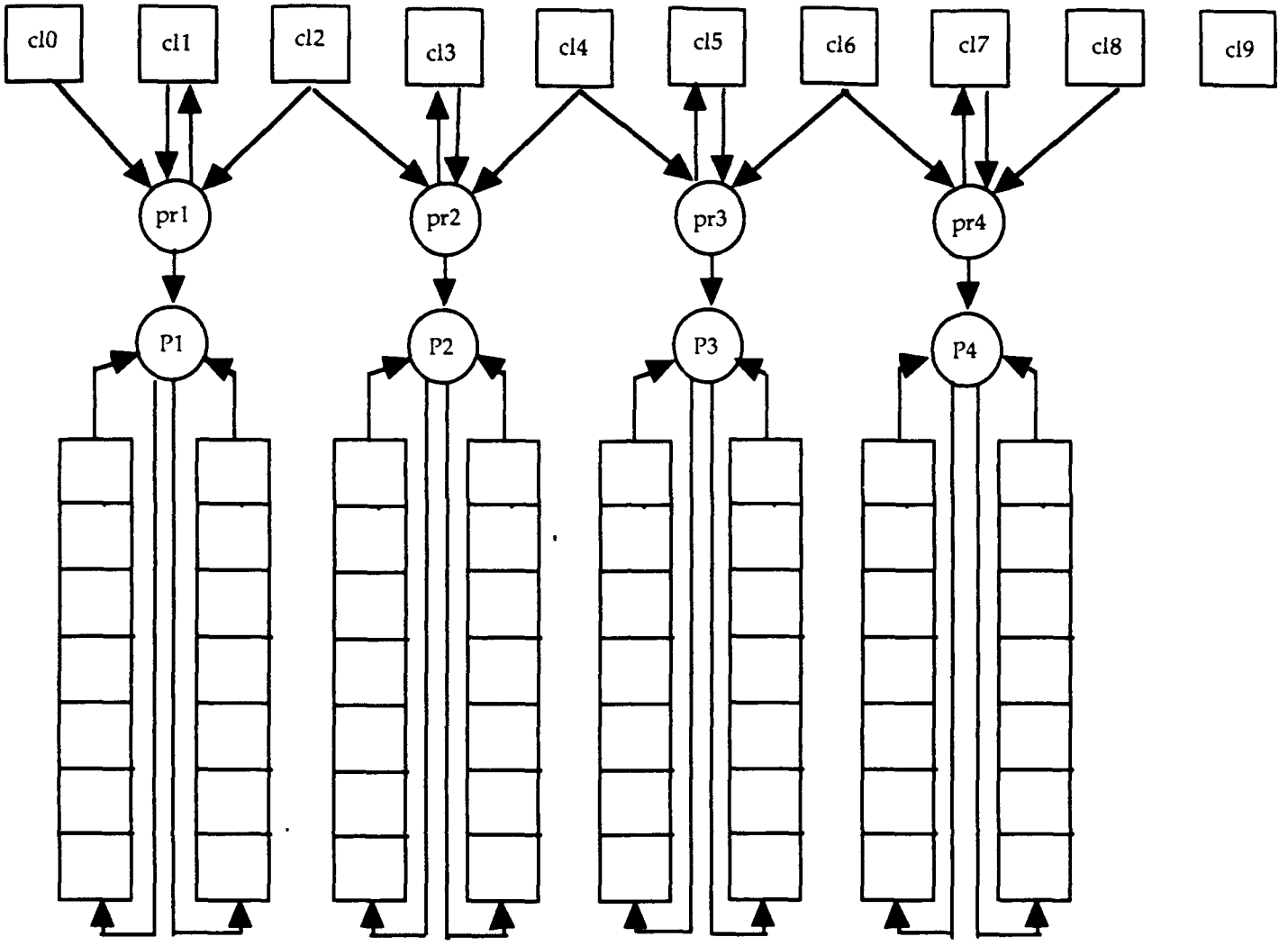
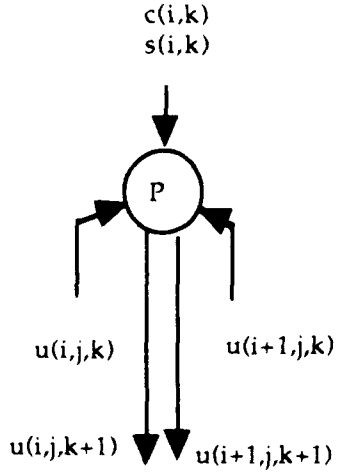


Figure.3.6 (a) Updating the eigenvectors during an odd step



$$u(i,j,k+1) = c(i,k)\ u(i,j,k) + s^{*}(i,k)\ u(i+1,j,k)$$

$$u(i+1,j,k+1) = -s(i,k)\ u(i,j,k) + c(i,k)\ u(i+1,j,k)$$

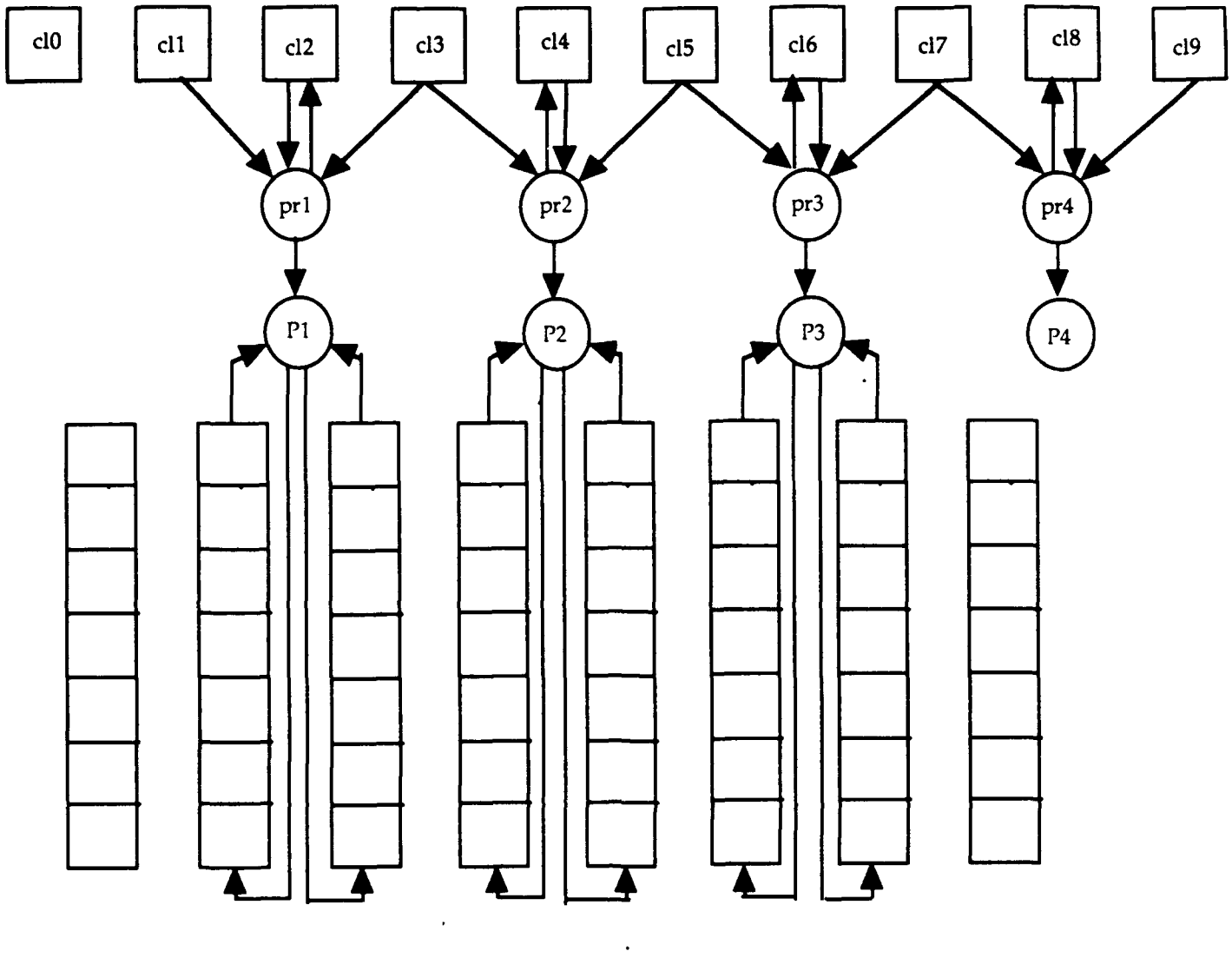


Figure.3.6 (b). Updating the eigenvectors during an even step.

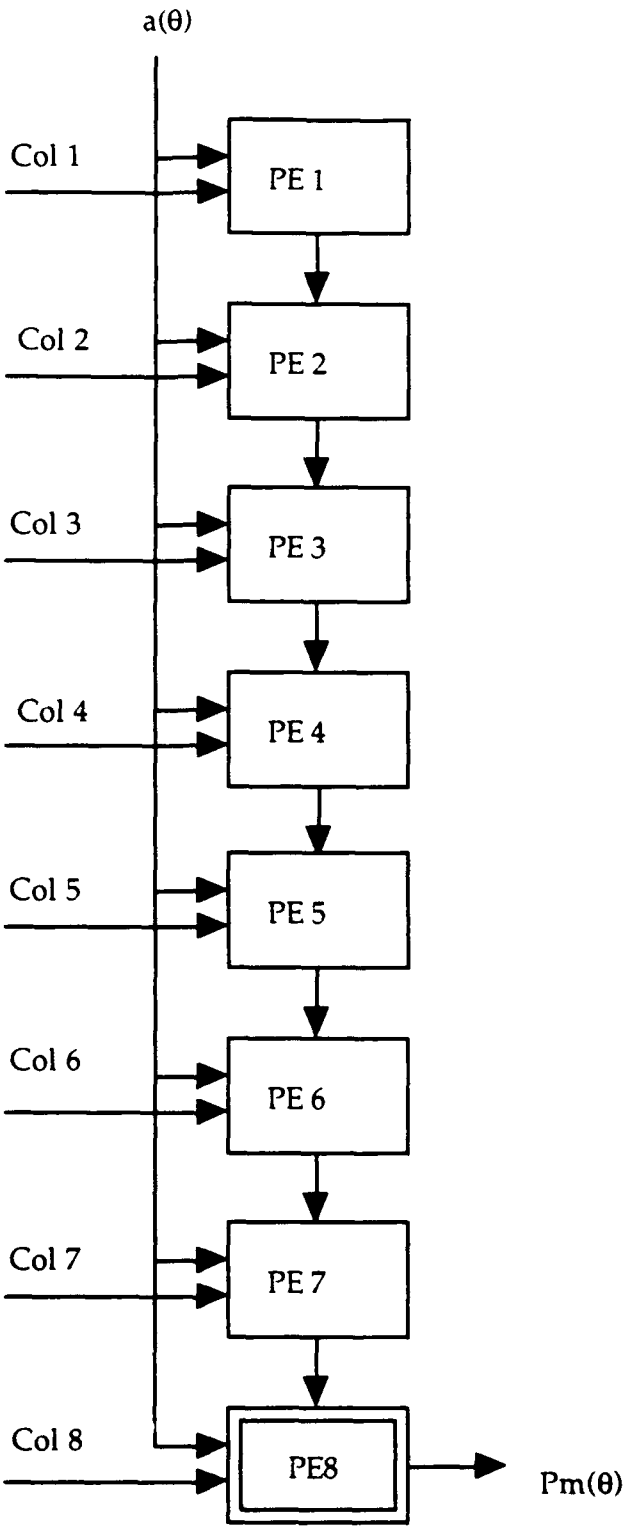


Figure 3.7: Hardware For Power Method

Chapter 4

DEVELOPMENT OF ARCHITECTURE FOR MUSIC ALGORITHM AND GENERALIZED PROCESSING ELEMENT

4.1 PIPELINED STAGES AND BUFFERS IN MUSIC ALGORITHM

The MUSIC algorithm is executed using four pipelined stages namely the covariance matrix formation, Householder method, QR method and the power method as stated previously. These pipelined stages along with buffers are shown in Figure 4.1. The covariance matrix stage which is the first stage collects the sampled data from 8 different sensors and computes 8×8 covariance matrix. The covariance matrix has 36 processors, all of them perform the same operation in parallel. The Householder method which reduces the covariance matrix to a tridiagonal matrix forms the second stage. The data generated in first stage is pipelined to the second stage through eight FIFO buffers. These FIFO buffers store data for transfer to the Householder stage. The hardware for the Householder method has 9 processors. PE1 to PE8 perform computations on each column of the covariance matrix and PE9 is used to compute data required by other processors. The tridiagonal matrix is reduced to a diagonal one by QR method which forms the third stage. Data generated by PE1 of second stage is pipelined to PE1 of third stage through a two word deep FIFO. The third stage has 2 kinds of processors, PE1 which computes eigenvalues and PE11 to PE18, which compute eigenvectors. Each processing element PE11 to PE18 generates eight eigenvectors which form the rows of the 8×8 eigenvector matrix. Eigenvectors computed by the third stage are pipelined power method which forms the fourth stage. It has eight processors PE1 to PE8 and each processor works on the array manifold and the column of the

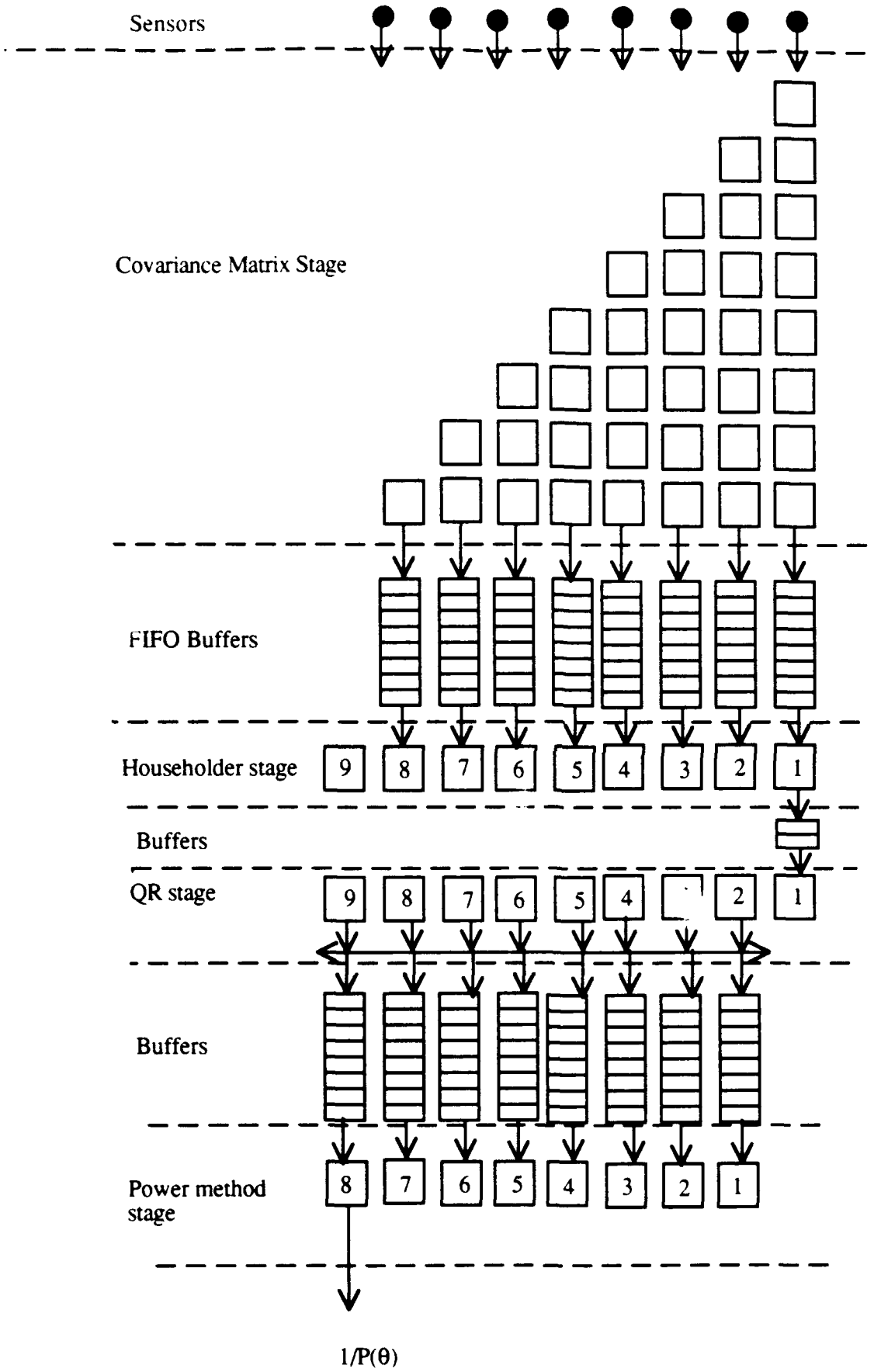


Figure 4.1: Various Stages and Buffers in MUSIC Algorithm

eigenvector matrix. Since QR method produces rows of the eigenvector an hardware switching method is used to convert rows to column before passing on to power method stage. Power method has 8 processorsthe eigenvector matrix needs to be output of PE8 gives the inverse of the power. This PE8 produces output for different angle of the array manifold. In the following section, a generalized PE architecture is described.

4.2 THE GENERALIZED PROCESSOR

As seen above there are four pipelined stages with eight different kinds of processing elements. All these processors perform similar arithmetic functions. Thus, various functions performed by different processors can be grouped together and performed by one generalized processor. Therefore instead of using eight different processors, a single generalized processor can be used.

The generalized processor named GP, has an architecture which is designed to maximize the throughput for different PEs required in the MUSIC algorithm. The internal architecture of GP has the capability to execute various operations in the MUSIC algorithm. After studying the capabilities required for the different PEs, a list of the following six arithmetic functions is compiled.

1) Addition
2) Subtraction
3) Multiplication
4) Division
5) Multiplication, Addition and Accumulation
6) Square rooting

Hence instead of using a generalized Arithmetic Logic Unit (ALU) as used in commercially available microprocessors, a block of three specified arithmetic functional units are utilized. These functional units are:

1) Addition/Subtraction Unit
2) Multiplication/Division Unit
3) Square Root Unit.

Computed data can be stored temporarily in embedde' 'andom Access Memory (RAM) or sent to the output port. The operation of data storing and I/O is performed using data bus.

The processor has eight, 16 bit general purpose registers. These registers are named as A, B, C, D, E and F. Each 16 bit register can also be used as two 8 bit registers in concatenation. It also has two index registers X and Y. Each index register is 8 bits wide. GP also has a 8 bit program counter (PC) and program status word register (PSW).

Thus the major components of the generalized processor GP are as follows:

1) Address bus
2) Data bus
3) Floating point adder/subtracter.
4) Floating point multiplier/divider.
5) Square rooter.
6) Embedded data memory.
7) Two Input ports.
8) Two Output ports.
9) Control unit

The above mentioned components along with the arrangement of the registers is shown in Figure 4.2 and each of them are explained as follows:

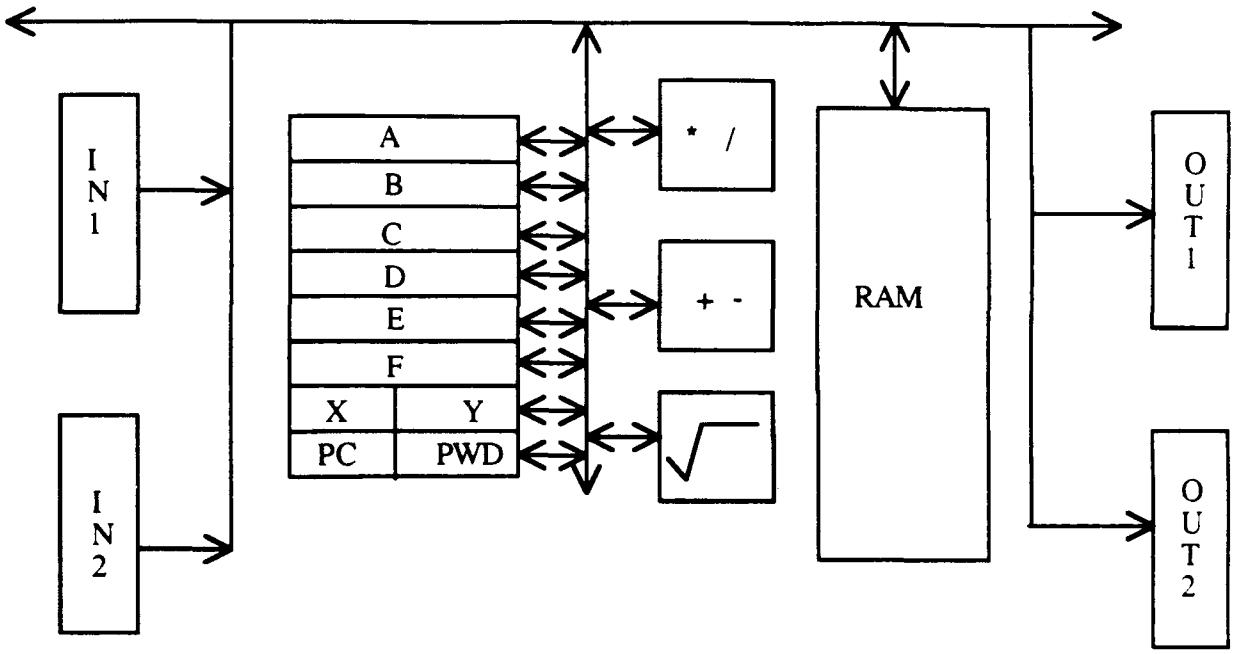


Figure 4.2 Architecture of Generilized Processor GP

Data and Address bus:

The data and the address bus are used for storing data in RAM and to transfer data through I/O port. The data movement in the chip occurs over a bi-directional 8 bit bus. The address is specified for internal data memory and I/O port by unidirectional address bus.

Floating point multiplier and divider:

GP has a dedicated multiplier/divider and is controlled by the control unit. The multiplier can perform 8 bit and 16 bit multiplications. The divider can perform 8 or 16 bit division (i.e. 8 or 16 bit divisor and 8 bit dividend). Since

multiplication and division are not performed in parallel in MUSIC algorithm, both of these operations can be performed by a single unit.

Floating point adder/subtracter:

Adder and subtractor can perform 8 and 16 bit addition or 8 and 16 bit subtraction. These various options are again specified by the control unit of GP. After studying different process requirements of the MUSIC algorithm it was unveiled that the addition and subtraction are not performed simultaneously in a single cycle. Hence one single unit can be used to perform addition and subtraction.

Square Rooter:

GP has a built in square rooter which performs square root operation on a 16 bit data and outputs 8 bit result. Since the square rooting is computed several times during the entire process, fast computation of square root is also required for high throughput.

Data Memory:

GP has a embedded RAM which is 8 bit wide and 256 byte long. It is mainly used to store data that is input from other processors or store data generated by the processor which is used for further computation.

Input/Output port:

The processor has two input and two output ports. Each of them is 8 bit wide and performs parallel data transfer. These input and output ports are memory mapped.

Control Unit:

The micro-programmed control unit generates control signals required for the operation of GP.

4.3 INSTRUCTION SET DETAIL

This section contains detailed information about each instruction in the GP instruction set.

Notation:

Each instruction description contains symbols used to abbreviate certain operands and operations. Table 1 lists the symbol used and their interpretation.

Table 1

D	Destination Operand
(D)	Contents of Destination Operand
M	Memory Location
(M)	Contents of Memory Location
opr	Operand
R	Any of the general purpose registers
(R)	Contents of the Register
S	Source Operand
(S)	Contents of Source Operand
$	Hexadecimal Number
#	Immediate Addressing Mode

Various instructions used are explained below. These instruction are discussed in alphabetical order.

1) Add

Operation S+D ⇒D

Syntax: ADD S,D

Description: Add source operand S to the destination operand D and store the result in destination operand.

2) Divide

Operation: S÷D ⇒D

Syntax: DIV S,D

Description: Divide source operand S by the destination operand D and store the result in destination operand.

3) Decrement

Operation R ⇐ (R) -$01

Syntax: DEC (opr)

Description: Subtract one from the contents of Register R.

4) Increment

Operation R ⇐ (R) +$01

Syntax: INC (opr)

Description: Add one to the contents of Register R.

5) Jump on Zero

Operation PC ⇐ Effective Address iff Z=1

Syntax: JZ (opr)

Description: Jump to the effective address if zero bit is set.

6) Jump on Non Zero

Operation PC ⇐ Effective Address iff Z=0

Syntax: JNZ (opr)

Description: Jump to the effective address if zero bit is not set.

7) Jump

Operation PC ⇐ Effective Address

Syntax: JMP (opr)

Description: Jump to the effective address.

8) Load

Operation D ⇐ (M)

Syntax: LD (opr)

Description: Load the contents of memory into destination register D.

9) Multiply

Operation S×D ⇒D

Syntax: MUL S,D

Description: Multiply source operand S by the destination operand D and store the result in destination operand.

10) Multiply Add Accumulate

Operation D+S1×S2 ⇒D

Syntax: MAC S1,S2,D

Description: Multiply source operand S1 by another source operand S2, Add to the destination operation D and store the result in the destination operand.

11) Negate

Operation R $\Leftarrow$ -(R)

Syntax: NEG (opr)

Description: Replace the contents of register R with two's complement.

12) Register Transfer

Operation D $\Leftarrow$ (S)

Syntax: TR (opr)

Description: Transfer the contents of source register S to the destination register D.

13) Square Root

Operation $\sqrt{S} \Rightarrow D$

Syntax: SQRT S,D

Description: Square root of source operand S and store the result in destination operand D.

14) Square

Operation $S \times S \Rightarrow D$

Syntax: SQR S,D

Description: Multiply source operand S to itself and store the result in the destination operand D

15) Store

Operation S ⇒ (M)

Syntax: ST (opr)

Description: Store the contents of source register S into the memory location.

16) Subtract

Operation S-D ⇒D

Syntax: SUB S,D

Description: Subtract source operand S from the destination operand D and store the result in the destination operand.

17) Square Add Accumulate

Operation D+S×S ⇒D

Syntax: SAC S,D

Description: Multiply source operand S to itself, Add to the destination operand D and store the result in destination operand.

This generalized processor is used for different computational modules of pipelined MUSIC algorithm. This processor GP as used as different PE are explained in detail. Programming details are also given.

4.4 PROGRAMMING DETAILS

Covariance Matrix

The covariance matrix is computed using 36 processors. Each processor inputs two samples from two different sensors. The values of the samples are multiplied added and accumulated for 4800 times. Once this process is completed, it is divided by the sample number (4800) to get the element of each covariance matrix. The computed elements of the covariance matrix are outputted to the set of buffers which form input to the Householders stage.

```
L1 LD    F     #4800   ;Load the counter to 4800 samples
   LD    AH    IN1     ;Input one of the values from IN1 port
   LD    AL    IN2     ;Input another value from IN2 port
   MAC   A             ;8 bit multiply add and accumulate in D
register
   DEC   F
   JZ    L1            ;Repeat the loop for 4800 times
   LD    A     #4800
   DIV   D,A           ;Divide D by A
   ST    D     OUT1    ;Output value of covariance matrix by OUT1
                       ;port
```

Householders Method

The Householder method has nine processing elements PE1 to PE9. The elements of the column from the covariance matrix form input to the eight processor - PE1 to PE8 of the Householders method.

PE1

PE1 computes diagonal and sub diagonal elements. One of the input and both the output ports of GP are utilized. The diagonal and the subdiagonal elements

computed are outputted to the next stage. The w and c computed are outputted to other processors of Householders method.

```
        LD    X     #00
        LD    FH    #7      ;Initialize the counter
L1      LD    AH    IN1     ;Load the elements of the column
        ST    AH,X          ;Store the values of column in memory
        ST    AH    OUT2    :Output the values of w7 to w2 to processor
2                           ;to 7 through OUT2 port
        DEC   FH
        JZ    L1            ;Repeat 7 times
L2      LD    AH,X          ;Load AH with w
        SAC   AH            ;Square w in order to get s
        DEC   FH            ;Repeat the process for 7 times
        JNZ   L2
        SQRT  D             ;Square root gives beta
        ADD   AH,DH         ; Add beta to r(1) to calculate w(1)
        ST    AH    OUT2    ;Send the value of w to other processors
                            ;through OUT2
        TR    DL,AL
        MUL   A             ;Mult beta with w(1)
        ADD   A,D           ;and Add it to r(1) to compute c
        NEG   DL            ;negate beta which is sub diagonal element
        ST    DL    OUT1    ;send it to next stage
```

PE2 TO PE8

The processors PE2 to PE8 are used to compute d and new values of the elements of the matrix. Two input and one output port of GP are used. One input is w

and c from PE1 and the other input is v from PE9. The d which is computed forms the output to PE9. PE2 to PE8 also compute new values of the elements of the matrix.

```
        LD    X     #00     ;Initialization X and Y
        LD    Y     #00
        LD    D     #0000
        LD    FH    #7      ; Initialization Counter
L1      LD    AL    IN1     ;Get in the value of r
        ST    AL    X+0     ;Store in memory
        LD    AH    IN2     ; Get value of w from PE1
        ST    AH    X+8     ; Store in the memory
        INC   X             ;Repeat for 7 times
        DEC   FH
        JNZ   L1
        LD    AH    IN2     ;Get value of c from PE1
        ST    AH    X+8     ;Store in memory
        LD    AH    Y       ;Load r
        LD    AL    Y+8     ;Load w
        MAC   A             ;Mult add and accumulate w and r =d
        DEC   FH
        JNZ   L2            ;Do for 7 times
        LD    CH    Y+15    ;Get the value of c
        DIV   D,CH          ;compute d=d/c
        ST    BL    OUT2    ;send output to PE9
        LD    FH    #7      ;Initialize counter =7
        LD    FL    #7
```

```
        LD    X     #00
        LDY   Y     #00
L3      LD    AH    IN1     ;get the values of v from PE9
        ST    AH    X+17    ;store in memory
        INC   X
        DEC   FH
        JNZ   L3
        LD    AH    Y+14    ;get fixed the value of w
        LD    BH    Y+22    ;get fixed value of v
        LD    BL    Y+8     ;get variable value of w
L4      LD    AL    Y+16    ;get variable value of v
        LD    CL    Y       ;load CL with r
        MULA                ;compute new value of elements of column
        MULB
        SUB   CL,AL
        SUB   CL,BL
        ST    CL,Y
        INC   Y             ;repeat seven times
        DEC   FL
        CMP   #00
        JNZ   L4
```

PE9

PE9 computes value of p and v. It utilizes 2 input and 1 output ports. One of the inputs is w and c from PE1 and the other input is d from PE2 to PE8. The v which is computed is sent to PE2 to PE8 through the output port.

```
        LD   X     #00     ;Initialize X and Y
        LD   Y     #00
        LD   FL    #7
        LD   FH    #7
L1      LD   AL    IN1     ;get the value of d from processors PE2 to PE8
        ST   AL    X
        LD   BL    IN2     ;bring in w
        ST   BL    X+8     ;store in the memory
        INC  X             ;repeat 7 times
        DEC  FL
        CMP  FL    #00
        JNZ  L1
        IN   BL    IN2     ;get the values of c
        SQR  BL            ;square c
L2      LD   AL    Y       ;load d
        LD   AH    Y+8     ;load w
        MACA               ;multiply add and accumulate d&w =p
        INC  Y
        DEC  FH            ;repeat 7 times
        CMP  FH    #00
        JNZ  L2
        DIV  BL,DL         ;p=p/c*c
        TR   DL,AL
        LD   AH,X+8
        MULA               ;multiply p and w
        LD   C,X
        SUB  C,A           ;compute v
```

```
        LD   CH   OUT1   ;output v to PE2 to PE8
```

QR Method

PE1

```
        LD   X    #00    ;Initialize
        LD   AH   X      ;Load value of Sine
        NEG  AH          ;Negate it
        LD   AL   X+2    ;Load value of b(sub diagonal element)
        MUL  A           ;Muliply b with -sin
        LD   BH   X+1    ;Load BH with Cos
        LD   BL   X+4    ;Load BL with a(diagonal element)
        MUL  B           ;Multiply a with Cos
        ADD  A,B         ;Add them to compute r
        TR   A,C
        SQR  A           ;Compute r*r
        LD   BH   X+3    ;Load BH with b
        SQR  BH          ;Square b
        ADD  A,B         ;Add them together
        SQRT A
        CMP  AL,#00
        JGT  L1
        LD   DH   #00
        LD   DL   #01
        JMP  L2
L1      DIV  AL.C
        TR   AL,DL
        LD   CH   X+4
```

```
        DIV   AL,DH
L2      ST    DH   X+6
        ST    DL   X+7
        TR    DL,BH
        TR    C,A
        MUL   A,B
        LD    BH   X+6
        LD    BL   X+3
        MUL   B
        ADD   A,B
        ST    A    X+8
        LD    AH   X+1
        LD    AL   X+7
        MUL   A
        LD    BH   X+3
        MUL   A,B
        LD    BH   X+6
        LD    BL   X+5
        MUL   B
        ADD   A,B
        ST    A    X+9
        LD    AH   X
        LD    AL   X+8
        MUL   A
        ST    A    X+2
        LD    AH   X+1
        LD    AL   X+7
```

```
MUL   A
LD    BH   X+6
LD    BL   X+9
MUL   B
ADD   A,B
```

4.5 CONCLUSIONS

Thus a generalized processor can be used as various processing elements for MUSIC algorithm. It can be programmed as shown above for different modules in MUSIC. Hence instead of designing different processing elements, a generalized processor can be used.

Chapter 5
DOA ESTIMATION FOR BROAD-BAND SOURCES USING "BROAD-BAND SIGNAL-SUBSPACE SPATIAL-SPECTRAL (BASS-ALE) ESTIMATION." ALGORITHM

5.1 INTRODUCTION

There are many Broad band DOA estimation algorithms available in the literature [26..27, 29, 36, 39..41]. Some of them are extensions of narrow band cases and others are transformed to specific Broad band algorithms. One of the broad-band methods proposed by Buckley and Griffiths [29] uses a focused covariance matrix as a temporal/spatial focused observation for broad-band source representation. A modal decomposition signal subspace algorithm has also been discussed by Su and Morf [27] and in their approach they decompose the rational spectra of the sources into elementary modes characterized by their poles.

In this chapter two algorithms for DOA estimation of Broad-Band signals are described. First, Coherent-Signal-Subspace processing proposed by Wang and Kaveh [40] is presented. The second algorithm proposed by Buckley and Griffiths namely "Broad-Band Signal-Subspace Spatial-Spectral (BASS-ALE) Estimation." is discussed in detail. An architecture for BASS-ALE algorithm is also presented.

5.2 "COHERENT SIGNAL-SUBSPACE PROCESSING FOR DETECTION AND ESTIMATION OF ANGLES OF ARRIVAL OF MULTIPLE WIDE-BAND SOURCES"

Wang and Kaveh proposed a model of M sensors organized in a geometry known to the processor. It is assumed that there are d signal sources that are stationary over the observation period and are represented by the vector

$$\mathbf{s}(t) = [\, s_1(t), s_2(t), \ldots, s_d(t) \,]^T \qquad (1)$$

Let $\mathbf{P}_s(f)$, a source spectral density, be an arbitrary dxd non-negative Hermitian matrix unknown to the processor. Also, let $\mathbf{P}_n(f)$, an MxM noise spectral density, be known except for a constant σ_n^2. Then, the array output $\mathbf{x}(t)_{Mx1}$ has a spectral density matrix

$$\mathbf{P}_x(f) = \mathbf{A}(f)\, \mathbf{P}_s(f)\, \mathbf{A}^{\dagger}(f) + \sigma_n^2 \mathbf{P}_n(f) \qquad (2)$$

where † denotes complex conjugate transpose and $\mathbf{A}(f)$ is an Mxd transfer matrix of the source-sensor array system with respect to some chosen reference point. Furthermore, the number of sensors M is supposed to be greater than the number of signals d.

The array $\mathbf{x}(t)$ is resolved in the temporal domain into non-superimposed narrow-band elements by using the Discrete Fourier Transform (DFT). The decomposed components are uncorrelated and the covariance matrix for the component f_j can be expressed as

$$\mathrm{cov}(\mathbf{X}(f_j)) \cong \frac{1}{\Delta T} \mathbf{P}_x(f_j) = \frac{1}{\Delta T} \mathbf{A}(f_j)\, \mathbf{P}_s(f_j)\, \mathbf{A}^{\dagger}(f_j) + \frac{\sigma_n^2}{\Delta T} \mathbf{P}_n(f) \qquad j=1, \ldots, J \qquad (3)$$

where the array output $\mathbf{x}(t)$ is divided into K subintervals of duration ΔT snapshots.
It is asserted that it is possible to lump the signal subspaces at different frequencies to produce a single signal subspace and still be able to extrapolate the number of sources and angles of arrival. This can be done using a transformation matrix $\mathbf{T}(f_j)$.

$$\mathbf{T}(f_j)\, \mathbf{A}(f_j) = \mathbf{A}(f_0) \qquad j = 1, \ldots, J \qquad (4)$$

However, the construction of $\mathbf{T}(f_j)$ requires a knowledge of the unknown angles of arrival. Wang and Kaveh speculate that a knowledge of the neighborhoods of these angles is enough as a preliminary estimate. In Effect, coherent minimum Akaike Information Criteria is used to estimate the number of wide-band sources d where d is chosen to minimize the function

$$\text{AICE}(d) = -2 \log \left[\begin{array}{l} \text{maximum of the likelihood function} \\ \text{of the observation obtained} \\ \text{by changing the d free parameters} \\ \text{in the pre specified model} \end{array} \right] + 2d$$

$$= K\,(M-d) \log\left(\frac{a_0}{g_0}\right) + d\,(2M-d). \tag{5}$$

The peak positions of the spatial spectrum are then estimated using MUSIC

$$\hat{P}(\theta) = \frac{1}{A^{+}(f_0)\hat{E}_n\,\hat{E}_n^{+}A(f_0)} \tag{6}$$

where $\hat{E}_n$ is the estimated noise eigenvector and $\hat{E}_n^{\dagger}$ is its complex conjugate noise eigenvector. Those angles that yield peak positions are the angles of arrival.

Although CSS spatial-spectrum estimation is effective, as bandwidths of various sources increase and their locations deviate further from focus points, asymptotic peak bias can increase. Also, the focused spatial/temporal covariance matrix that is used in BASS-ALE is a generalization of the focused matrix used in CSS. The principle difference in the two approaches is source representation. For these reasons, the "Broad-Band Signals-Subspace Spatial-Spectrum (BASS-ALE) Estimation" by Buckley and Griffiths has been adopted in this research.

5.3.1 ARCHITECTURE FOR "BROAD-BAND SIGNAL-SUBSPACE SPATIAL-SPECTRAL (BASS-ALE) ESTIMATION." ALGORITHM

The approach used by Buckley and Griffiths [1] is termed broad-band signal-subspace spatial-spectral (BASS-ALE) estimation. BASS-ALE estimators employ the eigenstructure of a broad-band data covariance matrix and broad-band source models. This signal approach is justified by identifying the low-rank character of broad-band source observations, thereby demonstrating the possible existence of signal and noise-

only subspaces. A general model of a source's location vector as viewed by an array of M sensors is represented as

$$\alpha_\theta(\omega) = [a_{1,\theta}(\omega), a_{2,\theta}(\omega), ..., a_{M,\theta}(\omega)]^T \quad (7)$$

where the source is at an angular frequency ω and coming from a three dimensional source θ. However, if the geometry of the array of sensors is restricted to a linear, equally spaced orientation where the sensor elements are of pure propagation delay and the sources are considered far-field, then a location vector representation can take the form of

$$\alpha_\theta(\omega) = \frac{1}{\sqrt{K}}[e^{-j\omega\tau_{1,\theta}}, ..., e^{-j\omega\tau_{M,\theta}}]^T \quad (8)$$

where $\tau_{i,\theta} = (i-1)\tau_\theta$. $\tau_\theta = (\Delta/c)\sin\theta$

θ is the azimuth angle measured relative to array broad side

Δ is the sensor spacing

c is propagation velocity

τ_θ is the propagation delay from the array origin to the ith sensor for the source location θ.

A location vector is a rank-1 model in that it spans a one-dimensional subspace in the M-dimensional observation space.

The location vector modeled above is used in the narrow-band spectral analysis. In broad-band, the angular frequency ω is no longer a constant. Consider the case where more than one set of (ω,θ) yields the same constant $\omega.\theta$, then those pairs have the same location vector representation. In that sense, the sets are ambiguous. By adding time delays into the picture, observations can be further separated by angle and can be more easily distinguished. Thus, in this method, both spatial as well as temporal source location modeling is required. If L delayed outputs are used, the location vector becomes ML-dimensional and it has the form of

$$\mathbf{a}_\theta(\omega) = \frac{1}{\sqrt{L}}[1, e^{-\omega T_d}, \ldots, e^{-j\omega(L-1)T_d}]^T \otimes \alpha_\theta(\omega) \quad (9)$$

where T_d is the temporal sample delay and $\otimes$ is the Kronecker product.

5.3.2 SIGNAL AND NOISE-ONLY SUBSPACES

It is possible to construct a ML-dimensional broad-band spatial/temporal vector x(n) by stacking L M-dimensional vectors $\chi(n - lT_d)$; l = 0, ..., L-1. The data consists of superimposed sources observed in a medium corrupted with additive noise. Assuming that the noise spatial/temporal covariance $\mathbf{R_n}$ is known, the broad-band data spatial/temporal covariance matrix is of the form

$$\mathbf{R_x} = \mathbf{R_s} + \sigma_n^2 \mathbf{R_n} \quad (10)$$

where $\mathbf{R_s}$ is the source only covariance matrix. Let the eigenvalues λ_i be ordered in descending order and let $\mathbf{e}_i$ be the corresponding orthonormal eigenvectors. Those λ_i's with values equal to σ_n^2 are the smallest and help determine the dimension of the space. If, for a given $\mathbf{R_x}$, there are D "significant" eigenvalues ($\lambda_i > \sigma_n^2$), then the effective dimension of the span of the space is D. Two eigenvector matrices $\mathbf{E_s}$ and $\mathbf{E_n}$ can be defined as such

$$\mathbf{E_s} = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_D] \quad (11)$$

and

$$\mathbf{E_n} = [\mathbf{e}_{D+1}, \mathbf{e}_{D+2}, \ldots, \mathbf{e}_{ML}] \quad (12)$$

The column spaces of these two matrices are the signal and noise-only subspaces of the ML-dimensional broad-band observation space. The temporal sample delay T_d has to be selected so as to adhere to single-channel Nyquist sampling. For d sources arriving from locations $\{\theta_k; k=1, 2, \ldots, d\}$ and for a given number of sensors M and a number of delay levels L, it is required that

$$d < \frac{ML}{(M + L)} \tag{13}$$

5.3.3 SPATIAL/TEMPORAL NOISE DECORRELATING

Assuming that the general $\mathbf{R}_n$ covariance matrix characterized by (4) is known, then it can be rewritten as

$$\mathbf{R}_x = \mathbf{R}_n^{-1/2} \mathbf{R}_x \mathbf{R}_n^{-1/2}$$

$$\mathbf{R}_x = \mathbf{R}_n^{1/2} [\, \mathbf{R}_n^{-1/2} \mathbf{R}_s \mathbf{R}_n^{-1/2} + \sigma_n^2 \mathbf{I}_{KL}] \, \mathbf{R}_n^{1/2}$$

$$\mathbf{R}_x = \mathbf{R}_s + \sigma_n^2 \mathbf{I}_{KL} \tag{14}$$

where "S" denotes decorrelated. Some of the sources are altered by this transformation and the location vector now becomes

$$a_\theta(\omega) = R_n^{-1/2} a_\theta(\omega) \tag{15}$$

The effective source dimension is now determined from the source sample covariance matrix constructed with the above mentioned location vectors.

5.3.4 THE BASIC BASS-ALE ESTIMATOR

Consider a spatial/temporal covariance matrix that has been estimated and that its dimension D has also been estimated. In narrow-band, a spectrum is estimated, for each location θ of interest, by comparison of the modulus squared of the scalar projection of the rank-1 location vector model onto one of the subspaces spanned by the eigenvectors of the covariance matrix. In broad-band however, a location vector-based estimator is suggested in which for each location θ, projections of vectors in the set of

location vectors $\{\mathbf{a}_\theta(\omega)\}$ are simply computed and combined. The location vector estimator is

$$\mathbf{P}(\theta) = \sum_{j=1}^{N_f} \frac{1}{|\, \mathbf{a}^{\dagger}(\omega_j)\, \hat{\mathbf{E}}_n \,|^2} \tag{16}$$

where N_f controls the sample density of the location vector set. Increasing N_f reduces sensitivity to source spectral extent. Good results can be obtained with small N_f.

5.3.5 BROAD-BAND COVARIANCE MATRIX ESTIMATION

Estimating $\mathbf{R}_x$ in the time domain can be directly derived from the array broad-band snapshot vectors. Let the set of K-dimensional independent broad-band snapshot vectors $\{\, \chi^T(nT_d);\ n = 1, 2, \ldots, N_t\}$ be the available data where N_t implies total. Furthermore, let

$$x(nT_d) = [\chi^T(nT_d), \chi^T([n-1]T_d), \ldots, \chi^T([n-L+1]T_d)]^T \tag{17}$$

be a KL-dimensional observation. Obviously, we can form sample covariance matrices as

$$\hat{\mathbf{R}}_x = \frac{L}{N_t} \sum_{n=1}^{N_t/L} x(nT_d)x^{\dagger}(nT_d) \tag{18}$$

Equation (12) is the average taken over independent vectors.

5.3.6 SIGNAL-SUBSPACE ORDER ESTIMATION

The covariance matrix is computed from broad-band spatial/temporal data corrupted with noise interference. $\mathbf{R}_s$ characterized in Equation (4) is rank-deficient and to find its dimension, the Akaike Information Criteria (AICE) is used.

For proposed dimension d, the total number of signal-subspace parameters to be estimated is

$$D(2ML - D) + 1. \tag{19}$$

Using a source representation subspace model, let

$$\mathbf{R}_x = \mathbf{R}_n^{1/2} [\, \mathbf{R}_n^{-1/2} \mathbf{R}_s \mathbf{R}_n^{-1/2} + \sigma_n^2 \mathbf{I}_{KL}]\, \mathbf{R}_n^{1/2} \tag{20}$$

where $\mathbf{R}_s = \mathbf{V}\mathbf{P}_s\mathbf{V}^\dagger$ and V is the ML-dimensional matrix whose columns are the source representation subspace basis vectors. The log-likelihood function is given by

$$L(D) = (J_2 - J_1 + 1)N(ML - D)\ln\left(\frac{a_0}{g_0}\right), \tag{21a}$$

where

$$a_0 = \frac{1}{ML - D} \sum_{i = D+1}^{ML} \hat{\lambda}_i, \tag{21b}$$

$$g_0 = \left(\prod_{i = D+1}^{ML} \hat{\lambda}_i \right)^{1/(ML - D)} \tag{21c}$$

and the $\hat{\lambda}_i$ are the decreasing eigenvalues of $\hat{\mathbf{R}}_x$. With (18), (20a-20c), select the signal subspace dimension $\hat{D}$ as the D that minimizes

$$AICE(D) = (J_2 - J_1 + 1)N(ML - D)\ln\left(\frac{a_0}{g_0}\right) + D(2ML - D). \tag{22}$$

A hierarchial structure of the method involved in Broad-Band Signal-Subspace Spatial-Spectrum (BASS-ALE) Estimation is shown in Figure 5.1. First, the covariance matrix of the collected data has to be estimated. Then the eigenvalues are computed using the Householder and QR methods. From the estimated eigenvalues, an estimation of the signal-subspace dimension D can be calculated according to the steps outlined above. Once the dimension of the system is known, the signal and noise-only eigenvectors can be constructed. The power method is used to find the desired locations θ using the location vector-based estimator $a_\theta(\omega)$.

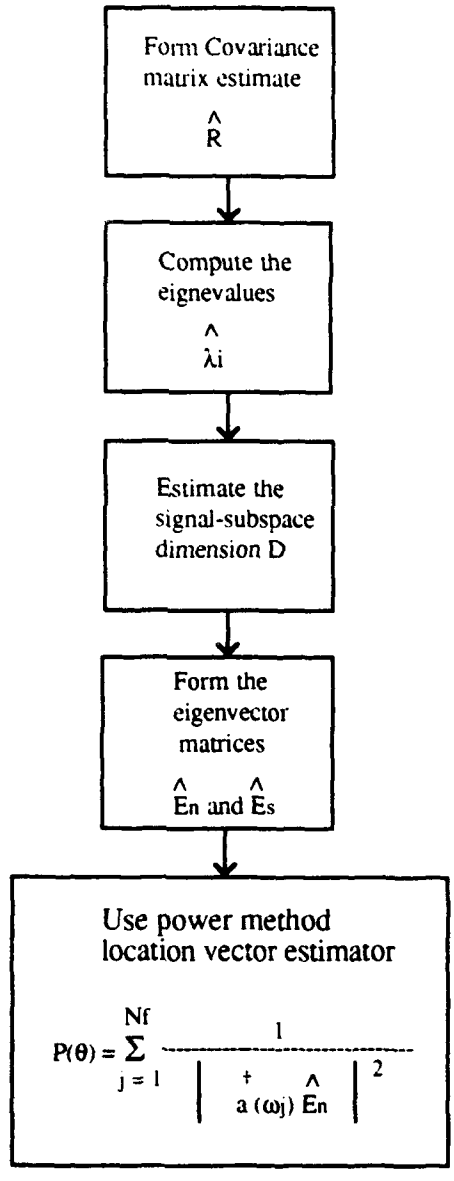
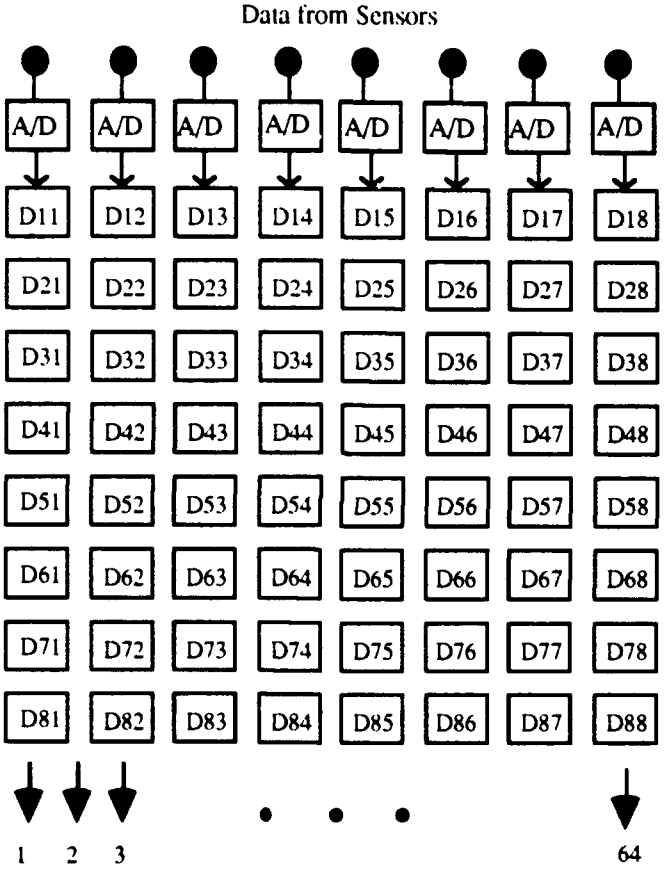


Figure 5.1 Procedural diagram showing the different units involved in the Broad-Band Signal-Subspace Spatial-Spectrum (BASS-ALE) Estimation.

5.4 ARCHITECTURE FOR BASS-ALE ALGORITHM

First of all the data need to be collected by the sensors to compute the covariance matrix . The data output from eight sensors is converted to the digital domain and fed to a pure propagation delay array in a parallel and pipelined fashion. The delay array is implemented using RAM for each sensor output below.



As in Equation (18), the data gathered in the delay array is collected once every eight cycles. In other words, the data is collected every time the array is filled with new vectors. The gathered data is stacked to construct a 64-element data vector required for the computation of the covariance matrix.

Computation of the covariance matrix involves a multiplication of 64 element vector with its 64 element complex conjugate transpose (64 element row) producing a 64x64 matrix for each set of data. Since the covariance matrix is symmetric, one way to reduce

the required number of computations is to compute only the lower triangular matrix. Figure 5.2 shows how a vector X(nTdL) (see equation 17) is multiplied by its conjugate transpose to obtain the lower triangular matrix. To compute the lower triangular matrix in parallel, it would require a total of 2080 PEs. On the other hand, if computation of one column at a time is performed in parallel, then 64 PEs can be used. Moreover, for every succeeding column, one of the PEs will be disabled; the last column to be computed will use only a single PE. This minimizes the amount of hardware used by a ratio of 2080:64 or 32.5:1, with the increase in computation time. The procedure's inefficiency increases with the increasing number of disabled PEs. Notice also that in each column, there is a common operand that is shared by all the elements of that column such as x_1^H in column 1 in Figure 5.2. These common operands are supplied as the broadcast elements. The second operand will be common in the row; e.g. x_{64} in row 64 in Figure 5.2.

5.4.1 BROAD-BAND COVARIANCE MATRIX ESTIMATION ARCHITECTURE USING 64 PROCESSING ELEMENTS

As the speeds required by the processing elements and the movement of data in the delay array are not the same, two clock frequencies are suggested. One frequency, adhering to Nyquist sampling, for collecting the data in the delay array, and another frequency to drive the processing elements. Figure 5.3 shows an architecture for computation of covariance matrix using 64 PEs. Data output from eight sensors propagate through a delay array, eight vectors deep. Once every eight cycles, the data is collected from the delay array to form a 64-element vector. An array of 64 PEs can be used to compute the lower triangular covariance matrix as shown in Figure 5.3. This array receives one set of operands from the sensors and the register file receives its conjugate transpose of that set of 64 elements. This vector is stored in the 64 PEs one element per PE and, simultaneously, the conjugate transpose of the vector is stored in the 64 registers file, one element per register. In this approach 64 multiplications will be

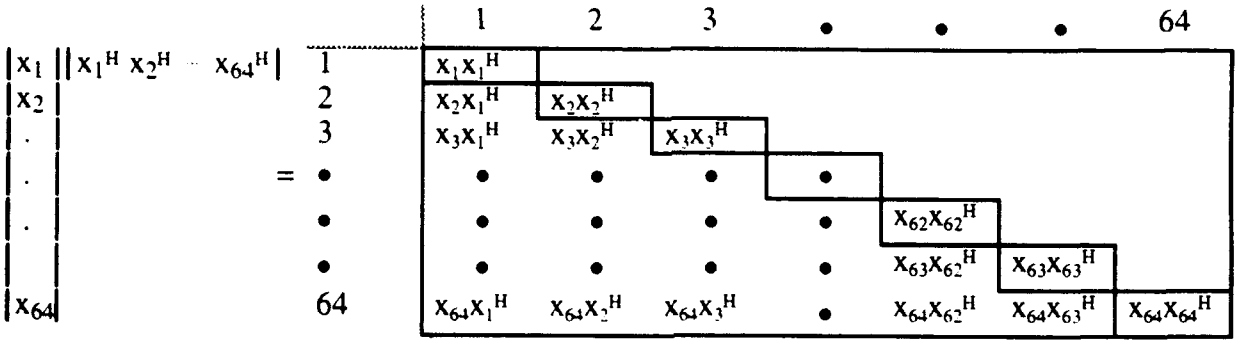


Figure 5.2 The product of a 64-element vector **x** by its conjugate transpose vector $\mathbf{x}^H$ resulting in a lower triangular matrix.

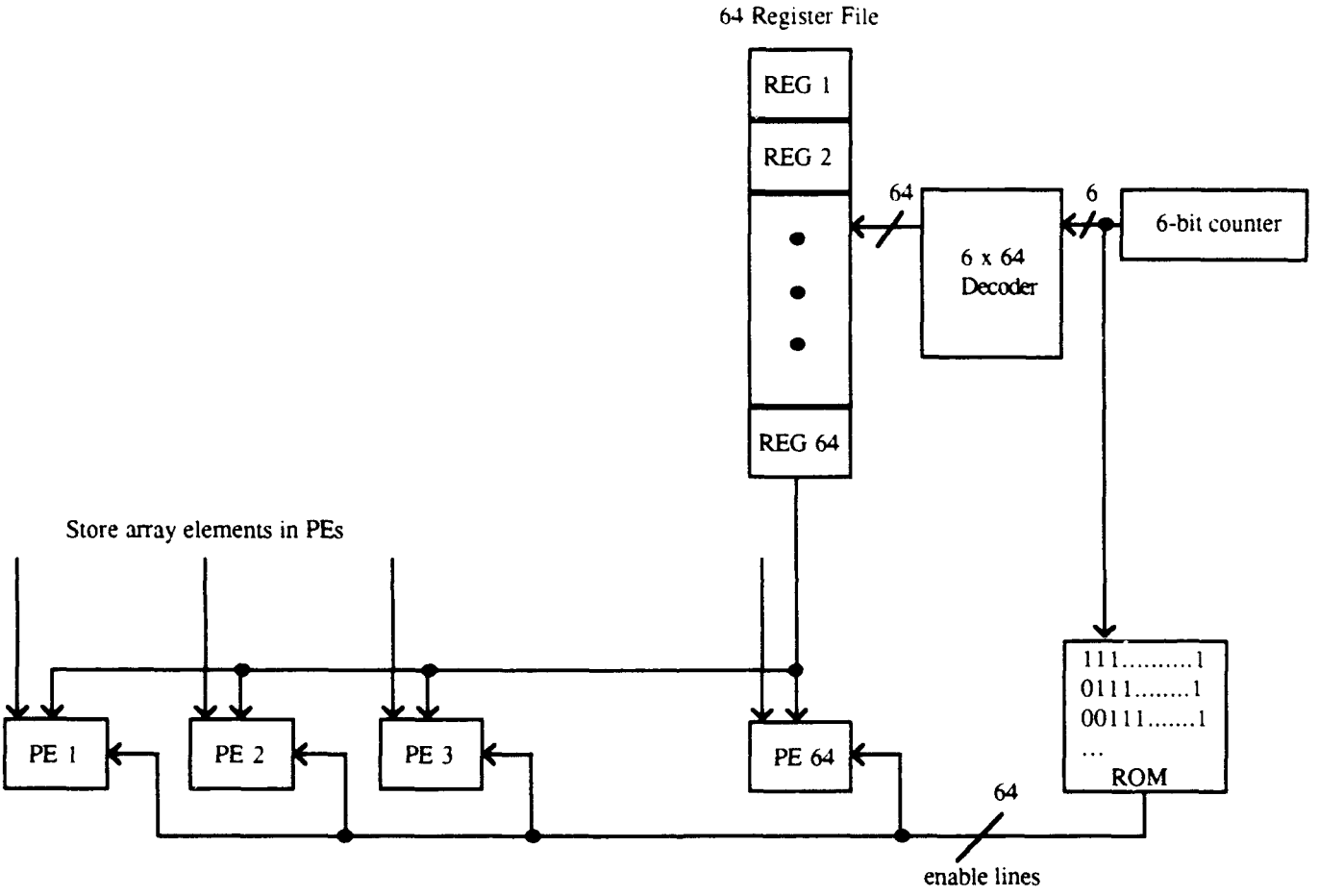


Figure 5.3 64 PE Architecture to produce a lower diagonal matrix without superimposed columns.

performed by 64 PEs in parallel. First operand is already stored in each PE and the second operand which is common for all is broadcasted from the register file. Again, the product is performed by producing one column of the covariance matrix at a time. All the enabled PEs work in parallel to compute that column. The 6-bit counter produces a sequence that is decoded (decoder not shown) to broadcast one operand from one of the registers at a time. The same counter sequence is used to access the micro code in the ROM. The micro code controls the operations of the PEs; it disables the PEs as needed. A total of 64 cycles are required to compute one frame of the covariance matrix. Moreover, all PEs are not used all the time. One approach to improve the computation time efficient use of PEs is developped and is explained in the next section.

5.4.2 BROAD-BAND COVARIANCE MATRIX ESTIMATION ARCHITECTURE USING 64 PROCESSING ELEMENTS (AN OVERLAPPED APPROACH)

In this approach, data can be overlapped by merging two non-full columns; e.g. column 2 and column 64 and is shown in Figure 5.4. This merging operation eliminates inefficient use of PEs. When two columns are incorporated, two operands are shared by each column; e.g. x_{63}^{H} and x_{3}^{H} in column 3 in Figure 5.4. When two columns are merged, two operands will be present; e.g. x_2 and x_{63} in row 2 of Figure 5.4. Therefore, every PE has at most two constants assigned to its first operand and a broadcast element for its second operand. Figure 5.5 shows a hardware configuration suitable to compute a lower triangular matrix with superimposed columns as previously discussed.

There are 64 PEs arranged as a linear array. On the bottom part of the circuit, three sets of registers are utilized; the left set consists of 32 registers (Set #1), the middle set consists of 31 registeres (Set #2), and the right set consists of only one register (Set #3). These three sets, as well as the set of 64 registers (top of Figure 5.5), get the data from a queue structure (the queue itself stores data output from the delay array and the

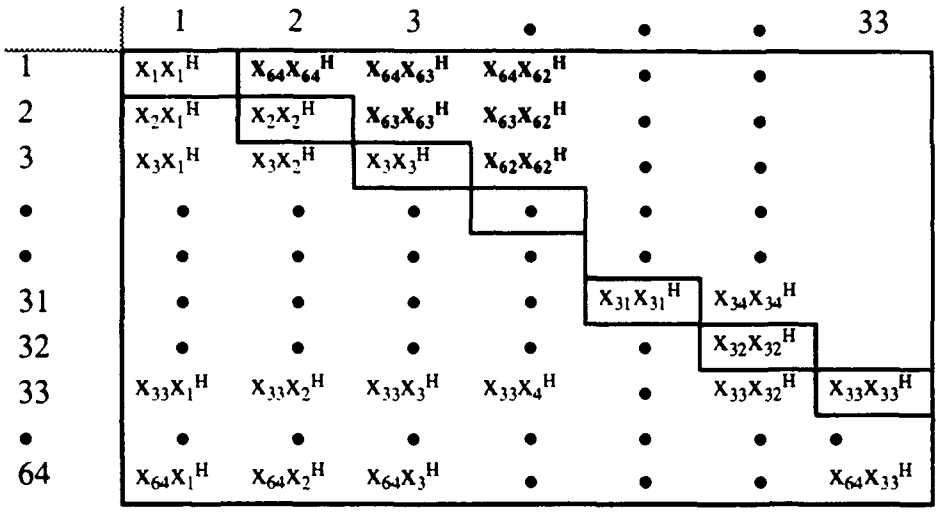


Figure 5.4 The product of a 64-element vector **x** by its conjugate transpose vector $\mathbf{x}^H$ resulting in a lower triangular matrix where its columns overlap.

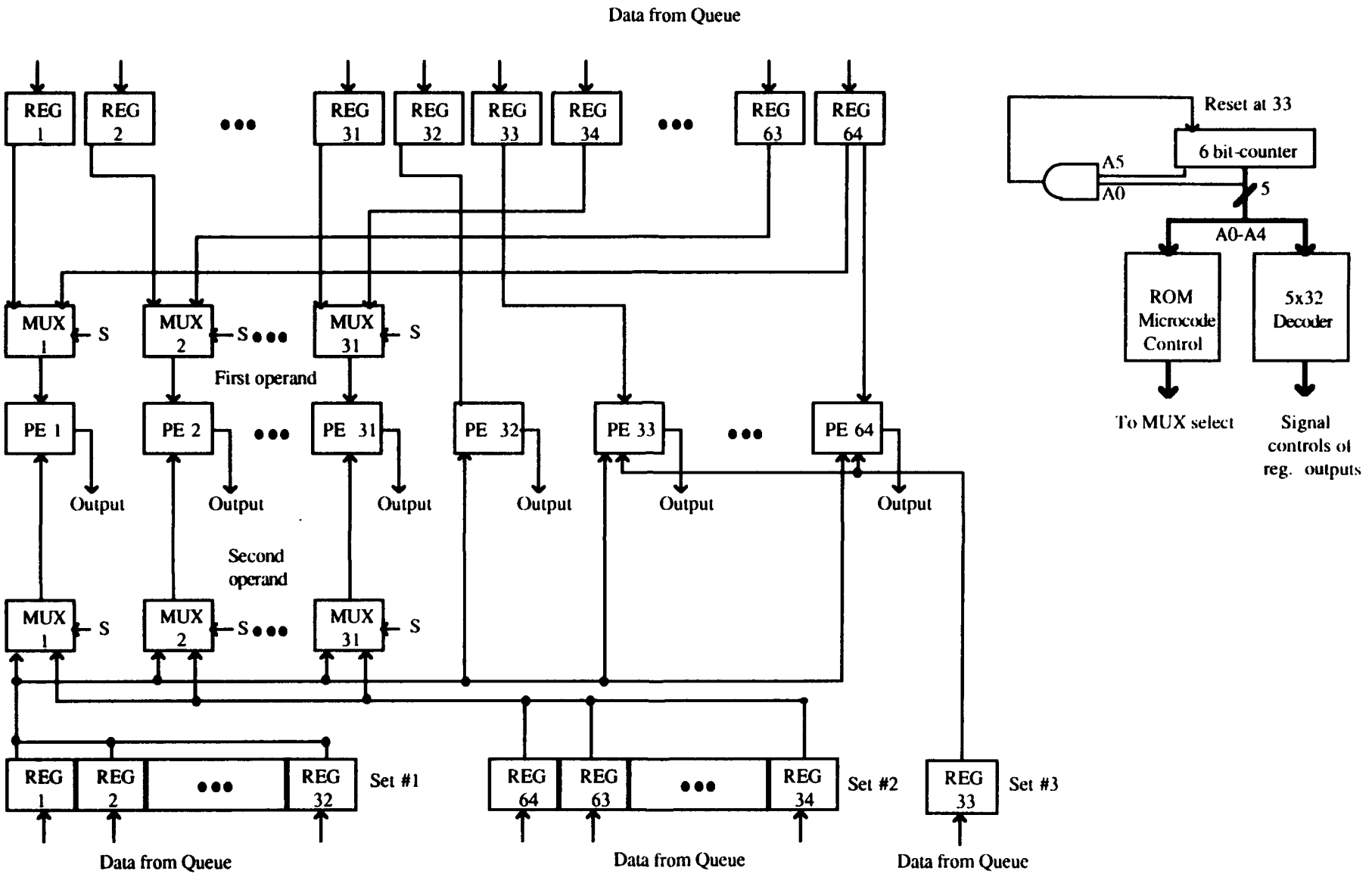


Figure 5.5 64 PE Architecture to produce a lower diagonal matrix with superimposed columns.

delay array in turn stores the data output from eight sensor elements array). Set #1 is connected to all of the processing elements. Set #2 is connected to the first 31 PEs and Set #3 is connected to the rest of the PEs starting from PE_{33}. There are two sets of 31 multiplexers. One set for the first operand and the second set for the second operand. Each PE has at the most two different values as a first operand such as x_1 and x_{64} for row 1 and x_2 and x_{63} for row 2. By the same contrast, each column receives at the most two different values as broadcast elements such as x_2^H and x_{64}^H for column 2 and x_3^H and x_{63}^H for column 3. The multiplexors are controlled by a micro coded ROM (31 bits wide) that is addressed by a six-bit counter. Address lines A0-A4 of the counter output are decoded and the 32 output lines of the decoder are used to control set #1, set #2 and set #3 register enable-output control lines. Using this method, one product $X(nTdL)X^H(nTdL)$ is achieved in 33 time steps. Since 4800 sensor sample readings are collected, this will require calculation of 600 such products. If the operating frequency is 100 kilohertz, it takes 80 microseconds (μs) to fill the delay array. It takes the architecture 1.28 milliseconds (ms) to finish its computations. As a consequence, a storage place to hold the arriving data from the sensors is required. For proper operation of the system, a queue structure, 454 64-element deep, can be used to balance the speeds of the sensors and the architecture.

Figure 5.6 shows the flow of the signals from the sensors, to the delay array, to the queue and multiplication unit, then to the accumulate unit. Previously calculated values already stored in memory are added to the column output of the multiplication unit. The multiplication and accumulation processes are repeated 33 times to produce one frame of the covariance matrix. A total of 600 frames are needed to produce the required matrix. The matrix elements are then divided by the number of iteration, here 600, to produce the covariance matrix. The divider unit is not shown.

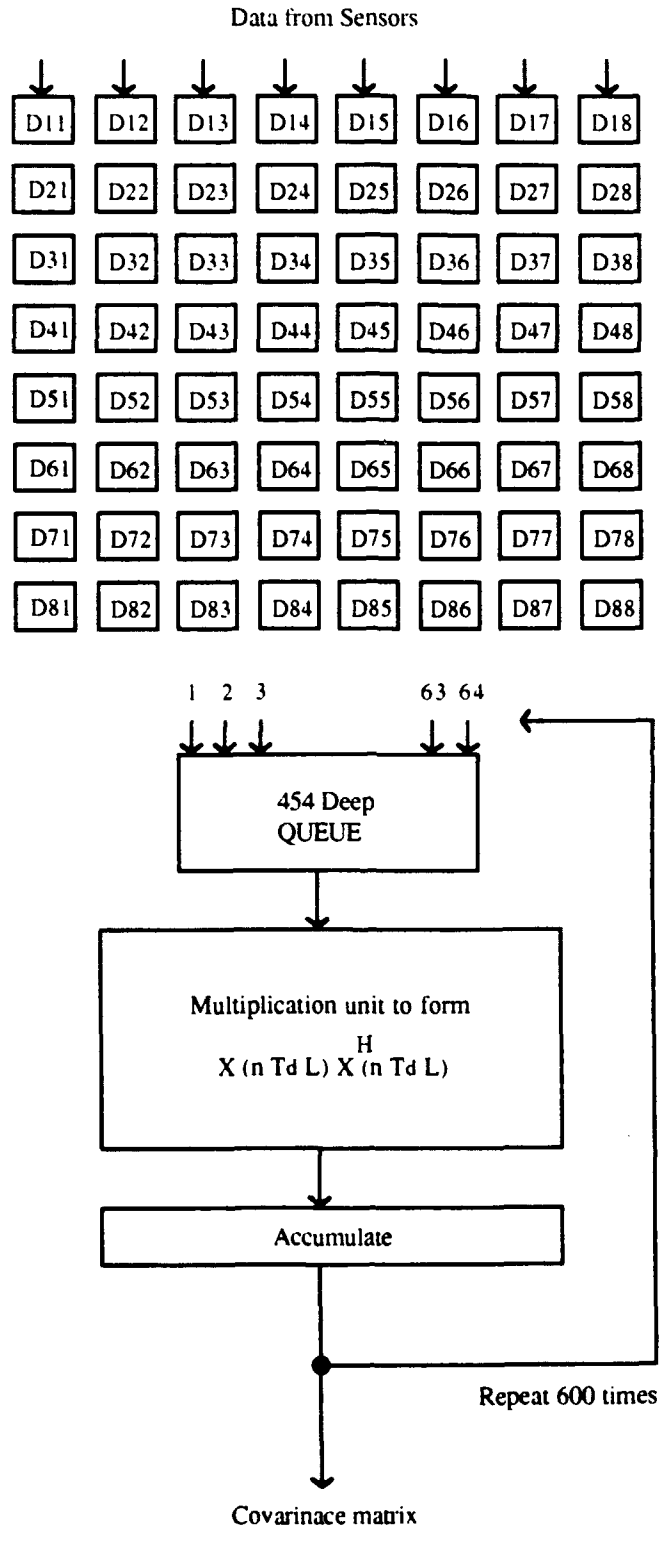


Figure 5.6 Flowchart of data from the delay array to the 64 PE multipilication unit.

Two architectures each using 64 Processing Elements (PE) are considered. The first approach requires less hardware, is simple to realize, but uses PEs inefficiently. The second overlapped approach requires a queue, multiplexers, and various sets of registers. The overlapped approach with its complex design reduces computation time. Nevertheless, it is still possible to compute the covariance matrix in a more elegant and efficient way by cutting down the number of processing elements used and increasing the number of computations. A new architecture is developed and is explained in the following section.

5.4.3.1 BROAD-BAND COVARIANCE MATRIX ESTIMATION ARCHITECTURE USING EIGHT PROCESSING ELEMENTS

A third approach of computing 64x64 covariance matrix is presented and uses eight processing elements. As described in previous sections, the elements of the delay array are stacked to create the 64-element data vector X(nTdL). To get more insight about the multiplication process of that vector with its conjugate transpose, a different representation of the data is adopted. The representation is demonstrated in Figure 5.7. Figure 5.7(a) illustrates how eight sub vectors, eight elements each, are stacked together to form a 64-element vector. The computation of the covariance matrix requires that this 64-element data vector (a column) be multiplied with its counterpart, the 64-element conjugate transpose data vector. The multiplication process creates a Hermitian matrix. Figure 5.7(b) depicts that lower triangular matrix in the form of sub vector multiplications. On account of creating a lower triangular matrix, 36 sub vector multiplications are required. Each of the sub vector multiplication produces an 8x8 sub matrix. One of these sub vector multiplications, namely $\tilde{x}(7)\tilde{x}(7)$, is taken as an example to illustrate the process. The resulting 8x8 sub matrix is shown in the zoom window. The complete lower triangular matrix is portrayed in Figure 5.7(c) where a

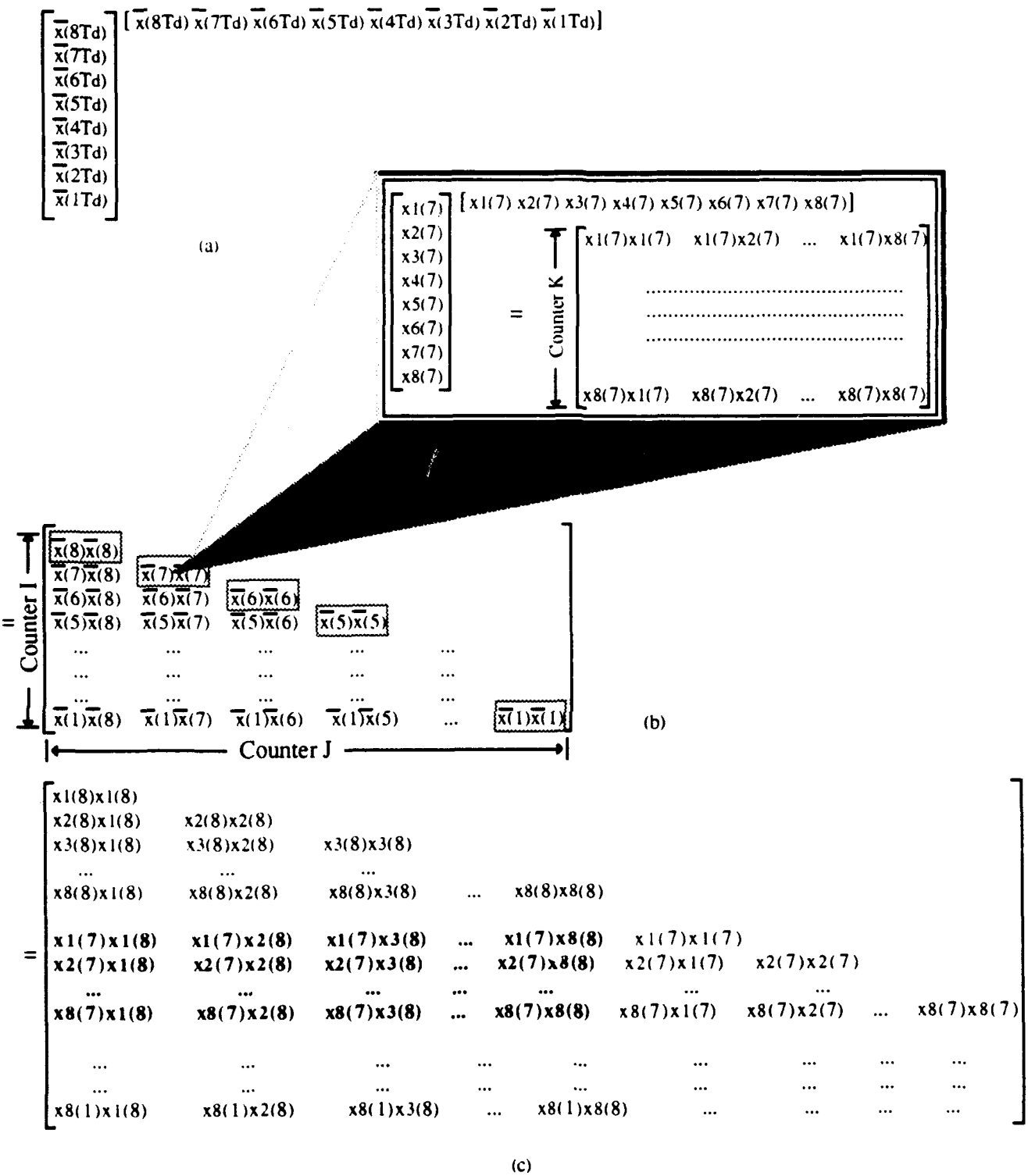


Figure 5.7 Vector X(nTdL) multiplication
(a) Vector X(nTdL) represented in a stack of 8 sub vectors.
(b) Product of sub vectors forming 36 sub matrices.
(c) Expansion of (b) to show the 8x8 sub matrices.

subscript denotes the position of that data element column-wise in the delay array and the index in parenthesis denotes the level of delay (or the row position). To distinguish between some of the sub vector multiplications, the results are portrayed with different highlights. One example is the sub-block represented in bold. Each row of that 8x8 sub matrix shares one of the operands as a constant; e.g., $x_1(7)$ in the first row, $x_2(7)$ in the second row and so on. Obviously, these values are the elements of the 7th delay row and they are broadcasted to be multiplied with the elements of the 8th delay row. This configuration has a close resemblance to the first approach design using 64 PEs. To simplify the computation of the covariance matrix, all the sub vectors will be computed in full including those that are on the diagonal. As a result, more data is generated than is desired, especially the ones above the diagonal. Indeed, the resulting matrix looks like descending stairs.

The addition of those extra elements to the matrix establishes a uniform algorithm where all the sub vectors can be multiplied in exactly the same manner without exceptions. In other words, one architecture can be used to compute the 8x8 sub matrices one at a time.

5.4.3.2 HARDWARE DESIGN

A block diagram of eight PEs that are needed to perform the task is shown in Figure 5.8. The hardware unit computes one of the sub matrices at a time. The broadcast data is stored in the registers and the second operand vector is stored in the PEs. An algorithm to compute the needed 36 sub vector multiplications is provided in the flowchart illustrated in Figure 5.9. Three counters are needed:

Counter J : Indexes the columns (A column refers to the different sub vectors of Figure 5.7(b))

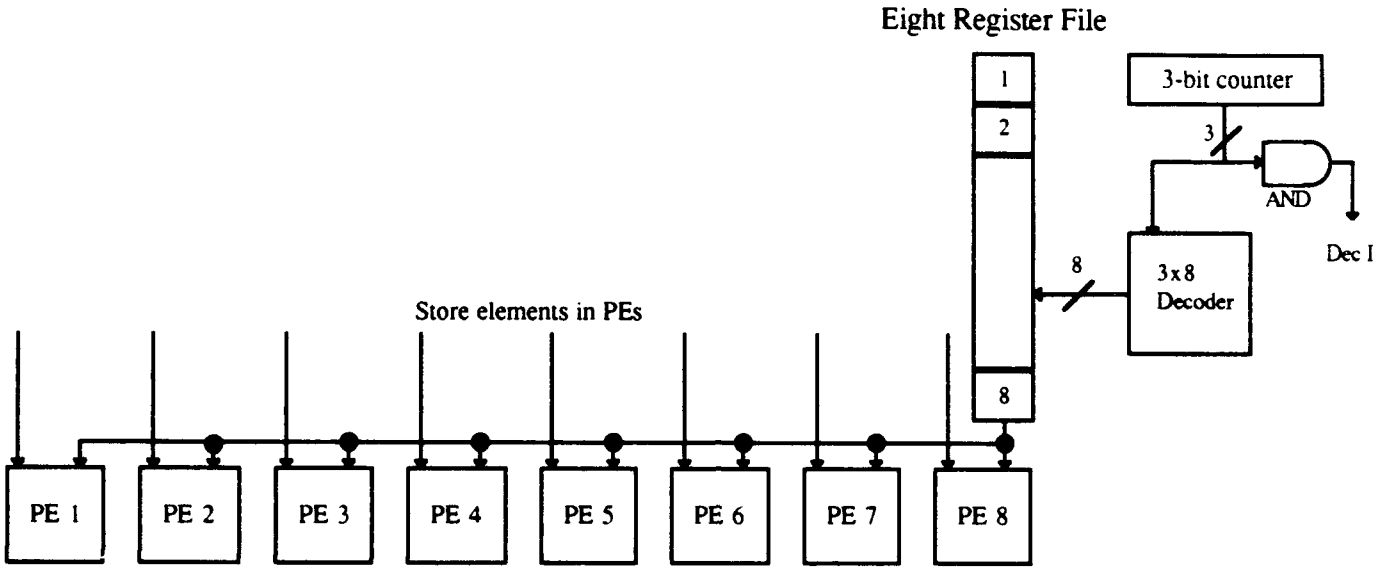


Figure 5.8 Block diagram overview of an 8 PE Architecture to compute a lower diagonal matrix.

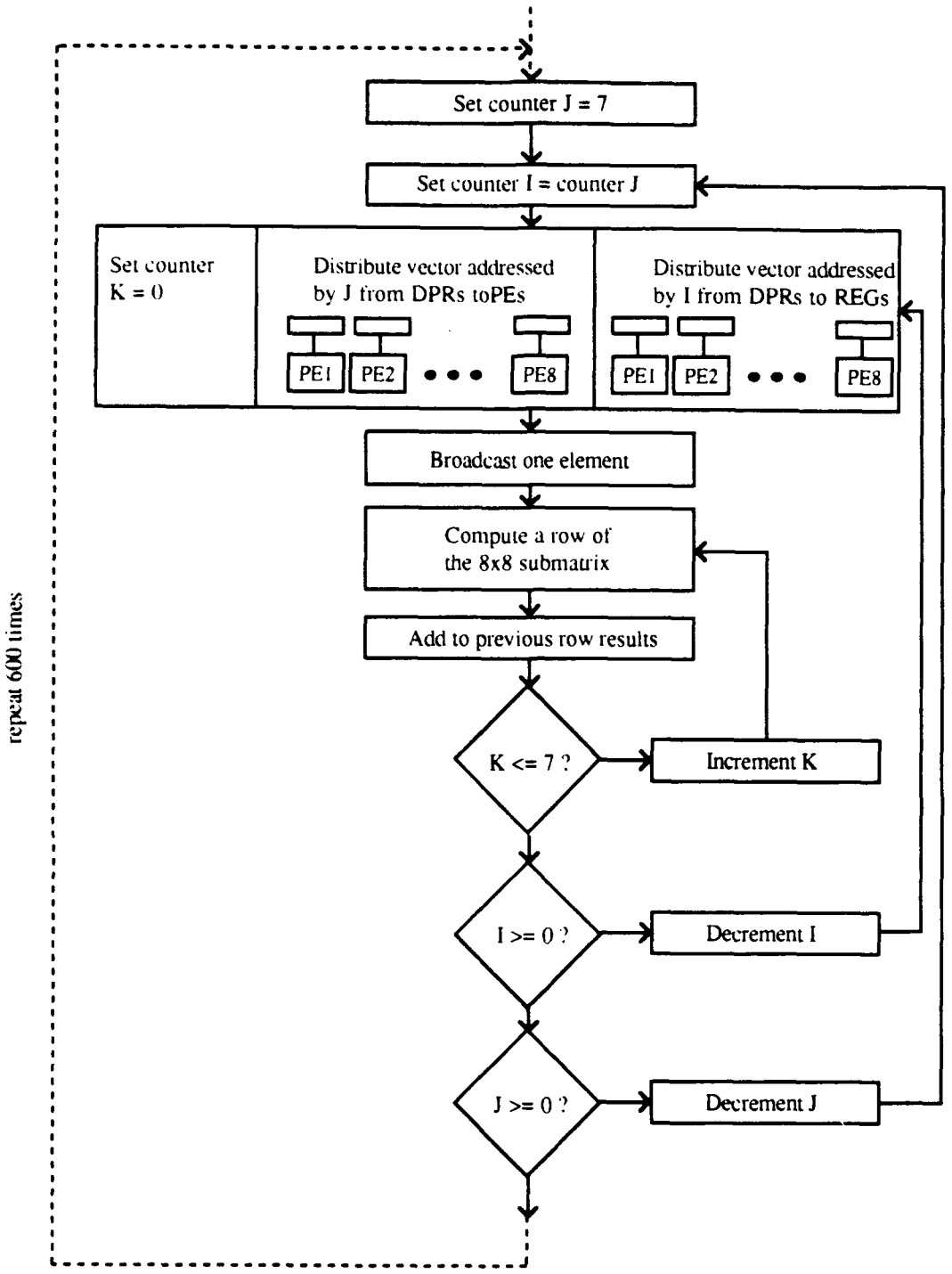


Figure 5.9 Flowchart illustrating the use of 3 counters to multiply 36 sub matrices forming a lower triangular matrix.

Counter I : Indexes the rows (A row refers to the different sub vectors of Figure 5.7(b))

Counter K : Indexes the rows within a sub matrix multiplication process.

The process proceeds as follows:

Set counter J = 7

Set Counter I = J = 7

All PEs compute in parallel

(a) One operand vector is stored in PEs and is specified by counter J

(b) Second operand vector is broadcasted (one element at a time specified by K) from a register file and is specified by I.

(c) Perform multiplication of one row in parallel

(d) Products are added to previously stored values.

Decrement the counter I until it reaches 0

Decrement the counter J and set I = J until counter J reaches 0

Repeat for 600 iterations.

Figure 5.10 shows the needed architecture to perform the operations explained above. In this design, the following hardware parts are used:

Sixteen	Dual-Port Rams (DPR) 8-words deep
Four	2 to 1 Multiplexors
Four	3-bit Counters
One	9-bit Counter
Eight	Registers, and
Eight	Processing Elements.

Data output from eight sensors are fed and written into eight dual-port RAMs. At any one time, one level of DPRs will be in write mode storing newly read data from the

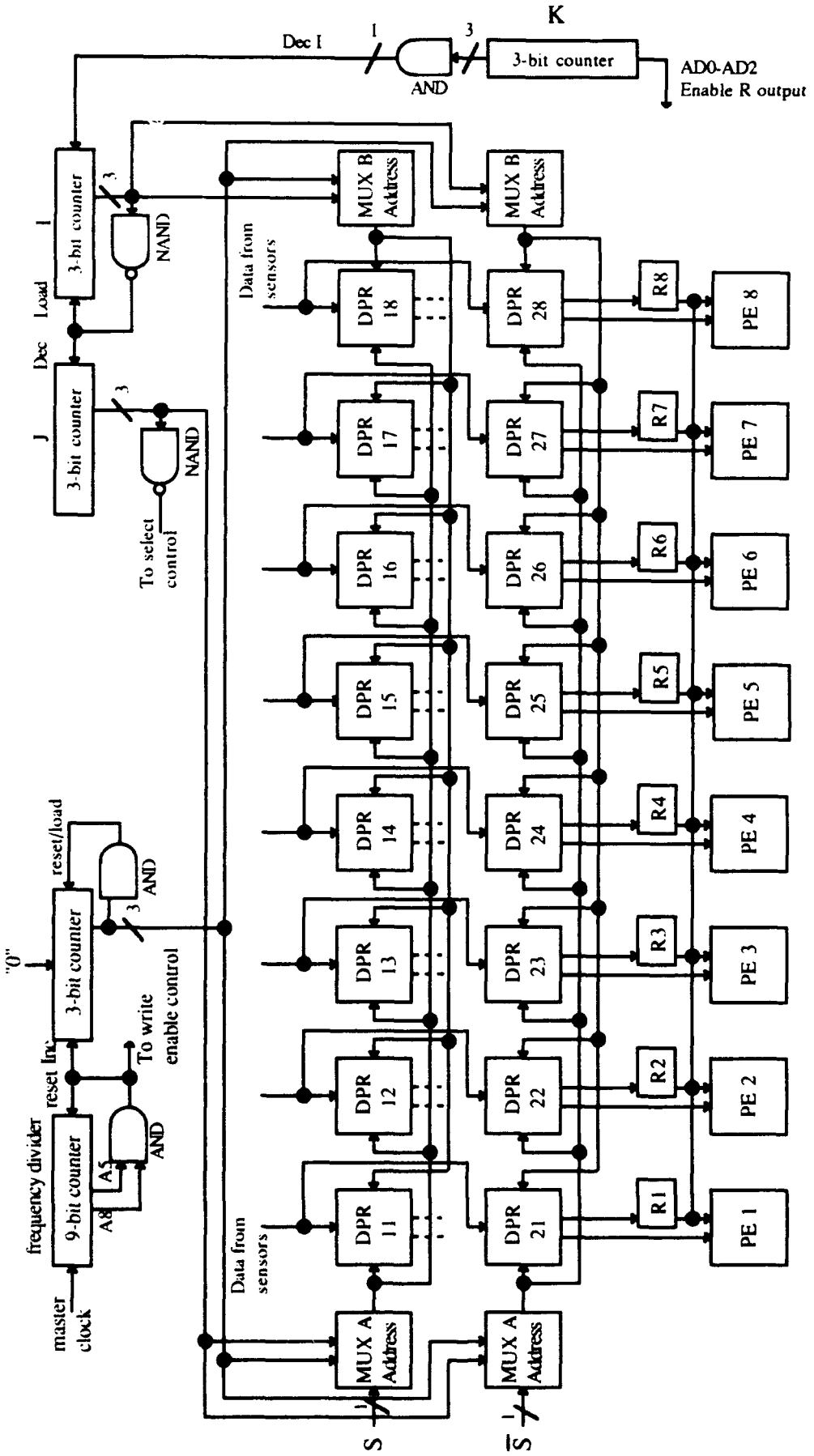


Figure 5.10 Detailed floor plan of 8 PE Architecture to produce a lower triangular matrix

sensors' output while the second level of DPRs will be in read mode where previously stored information is now being used to compute one vector product X(nTdL) with its conjugate transpose. Addresses needed by the DPRs are provided by counter I, counter J and a 3-bit counter which is controlled by a 9-bit counter. Two multiplexors, controlled by S (or $\bar{S}$), direct the needed address to the DPRs. If the DPRs are in write mode, the 3-bit address is selected, otherwise, address I and address J are selected. In read mode, the data addressed by counter I is supplied from each DPR to the respective register, while simultaneously the data addressed by counter J is supplied from each of the same DPRs to the respective processing element. Counter K is initialized to the value of zero and then is used to broadcast the output of one of the registers, one at a time, to all of the eight processing elements. Each PE then multiplies that register output data with the value already stored in its internal register (an element of vector J). The multiplication result is added to previously computed values that are stored in memory at an address pointed to by counters I, J and K. Counter K loops through its range (0→7) to construct an 8x8 sub matrix. Counters I and J loop through their range (7→0) to compute the 36 sub matrices. At the end of three loops (I, J, K), the assignment of the two levels of DPRs are switched and the operations performed by the PEs are repeated for the newly available data. The process is repeated 600 times to build the required matrix. The covariance matrix is formed by collecting the matrix and dividing its elements by 600.

Figure 5.11 shows the control lines for each Dual Port RAM (DPR). Since the architecture is designed to perform one sub matrix at a time, it is possible to detect when the PEs finish the computations of that sub matrix. At that time counter K reaches its highest state ($K=7_{10}=111_b$) . If at that time counter J is indexing the last column ($J=000_b$), the computation of the lower triangular matrix is complete. This logical function is implemented using two AND gates and one NAND gate. The output of the AND gate is used as a clock input to the D flip-flop. The value stored in the flip-flop is

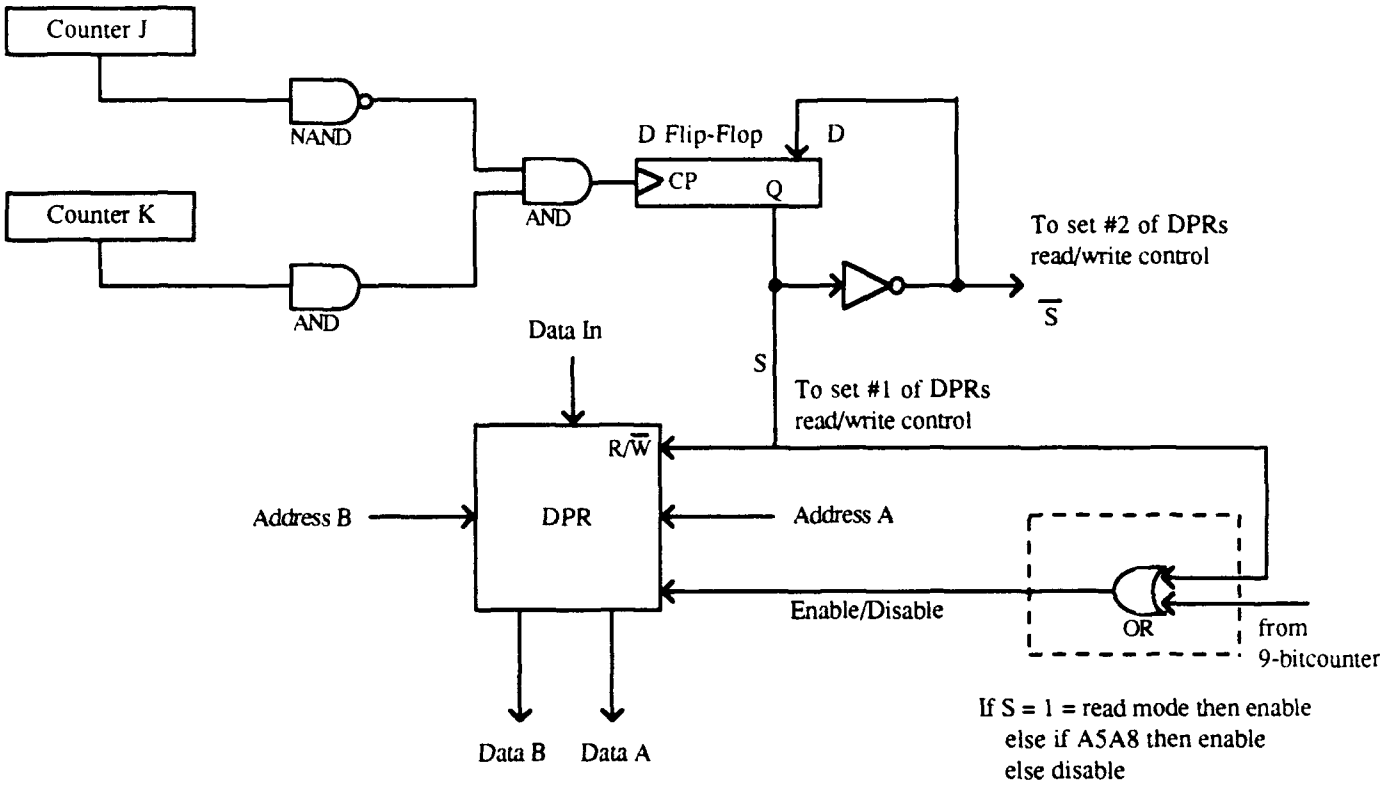


Figure 5.11 Dual Port RAM control block.

used to select one level of the DPRs; either the top level or the bottom level, while the inverted output $\bar{S}$ of the flip-flop is used to select the second level of the DPRs. The selection is meant to be used as a read/write control. Notice also, that there is an enable/disable control on a DPR. The only time that the chip should be enabled is when it is time to read information for processing (S=1), or when it is time to write the newly read sensor output data (S=0 write mode). An OR gate is used to enable the DPRs. When a DPR is in read mode, it receives address I and address J. On the other hand, when the DPR is in write mode, it receives its only address from a 3-bit counter controlled by the 9-bit counter (see Figure 5.10).

The internal structure of each processing element is shown in Figure 5.12. As illustrated, the following hardware elements are needed:

Four Multipliers

Four Adders

Three Registers

One of the values stored in the external registers is broadcasted to all the PEs. Complex multiplication of the value with the data stored in the internal register is performed. Four multipliers are needed due to the complex nature of the data. It is also desirable to use as many multipliers to generate parallelism to enhance speed of the system. Two adders are used to add the results of the complex multiplication; one adder for the real part and another for the imaginary part. Previously calculated data that has been stored in memory, is retrieved and fed through two sets of registers before it is added to the newly computed complex data. It is possible to eliminate the latter two registers to minimize the number of hardware needed in each PE provided that all the propagation delays have been accounted for in the multiplier then the adder stages.

CONCLUSION

In this chapter BASS-ALE algorithm has been simplified and converted into parallel/pipelined algorithm. First part of this algorithm requires computation of covariance matrix. Three architectures are proposed and an architecture with 8 PEs has been selected for detailed design. Design for the next module is in progress and other parts are being designed separately.

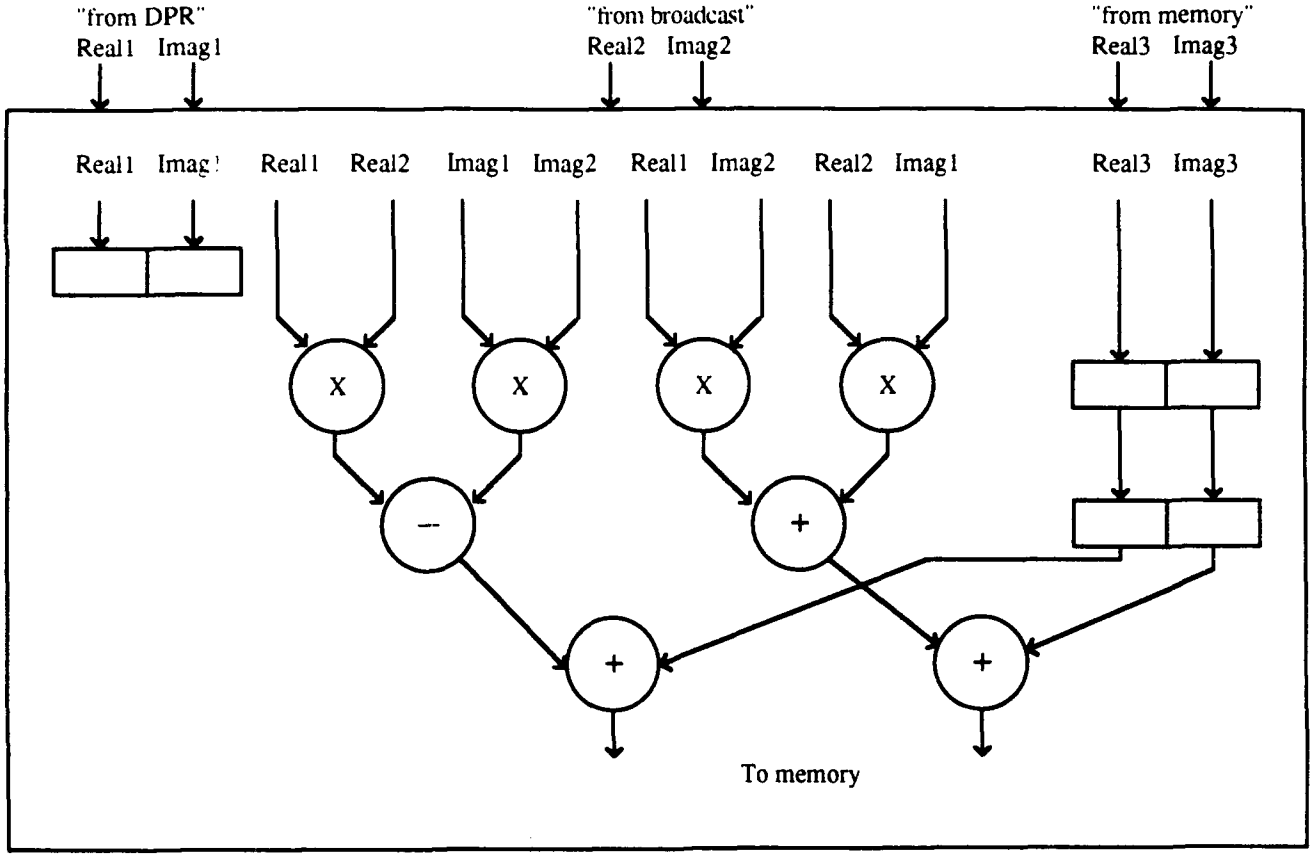


Figure 5.12 Detailed internal structure of a Processing Element.

Chapter 6

DOA ESTIMATION FOR BROADBAND SOURCES USING A BILINEAR TRANSFORMATION MATRIX APPROACH

6.1 INTRODUCTION

The estimation of angle of arrivals of multiple broadband sources has been carried out in a variety of ways over the past few years. The conventional approach is to form a generalized correlator [31] to estimate the Time Difference Of Arrival (TDOA) of the signal at the sensors. The so called maximum likelihood based methods [32-34] require knowledge of the source and noise spectra and are computationally expensive. The parameter estimation based methods [35-37] assume Auto-Regressive Moving Average (ARMA) models for the received signals and the estimated ARMA parameters are utilized for the TDOA calculations. Such model based methods have computational complexity and their effectiveness depends upon the approppriateness of the model chosen to represent the unknown broadband signals. Another way is to extend the ideas from the narrowband case [38] and use a eigendecomposition approach for the estimation. This approach involves the incoherent combination of the eigenvectors of the estimated spectral density matrices at each frequency bin to calculate the TDOAs. One other way [39,40] is to use the initial estimates of the angles of arrival to transform the eigenspaces at different frequency bins and generate a single coherent subspace which is eigendecomposed to give more accurate estimates. Well separated angles can be estimated by focusing at different angles at each time and iterating to obtain the accurate results.

Shaw and Kumaresan [26], proposed an algorithm for broadband DOA estimation using a simple bilinear transformation matrix. An approximation

resulting from a dense and equally spaced array structure is used to combine the individual narrowband frequency matrices for coherent processing. When compared to other coherent approaches, this algorithm has the advantages of being non-iterative and does not require any initial estimates of the angles of arrival and all angles are computed from a single step of coherent subspace calculations. Hence it was found to be a suitable algorithm for computation of DOA using dedicated hardware. The algorithm has been described in the following section.

6.1.1 Problem Formulation

Consider a linear array with 8 sensors which are spacedat equal distances. The incoming signal is assumed to be composed of d plane waves emitted from d sources ($d < 8$), with an overlapping bandwidth of B Hz. The signal from the kth sensor is expressed as

$$r_k(t) = \sum_{i=1}^{d} s_i\left(t - (k-1)\frac{\Delta}{c}\sin\theta_i\right) + n_k(t) \qquad (6.1)$$

$$-\frac{T}{2} \le t \le \frac{T}{2} \qquad 1 \le k \le 8$$

where $s_i(.)$ is the signal radiated by the ith source, Δ is the separation between the sensors, c is the propagation velocity of the signal wavefront, θ_i is the angle that the ith wavefront makes with the line of array and n_k is the additive noise at the ith sensor.

Performing the FFT and representing the two sides by their Fourier coeffficents

$$R_k(w_l) = \sum_{i=1}^{d} e^{-jw_l(k-1)\frac{\Delta}{c}\sin\theta_i} S_i(w_l) + N_k(w_l) \qquad (6.2)$$

with $w_l = \frac{2\pi}{T} l$, $l = l_1, \ldots, l_1+n_f$, where w_{l_1} and $w_{l_1+n_f}$ are the frequencies which span the bandwidth B.

Writing in the matrix notation

$$\mathbf{R}(w_l) = \mathbf{A}(w_l)\,\mathbf{S}(w_l) + \mathbf{N}(w_l) \qquad (6.3)$$

where these matrices are composed of the column vectors

$$\mathbf{R}(w_l) = [\mathbf{r}_1(w_l) \ldots \mathbf{r}_8(w_l)]^T \qquad (6.4a)$$

$$\mathbf{N}(w_l) = [\mathbf{n}_1(w_l) \ldots \mathbf{n}_8(w_l)]^T \qquad (6.4b)$$

$$\mathbf{S}(w_l) = [\mathbf{s}_1(w_l) \ldots \mathbf{s}_d(w_l)]^T \qquad (6.4c)$$

and the matrix $\mathbf{A}(w_l)$ is a $8 \times d$ direction finding matrix

$$\mathbf{A}(w_l) = \begin{bmatrix} 1 & \ldots & 1 \\ e^{-jw_l\tau_1} & \ldots & e^{-jw_d\tau_d} \\ \ldots & \ldots & \ldots \\ e^{-j7w_l\tau_1} & \ldots & e^{-j7w_d\tau_d} \end{bmatrix} \qquad (6.4d)$$

$$\tau_i = \frac{\Delta}{c}\sin\theta_i \qquad (6.4e)$$

τ_i being the TDOA of the ith source. Assuming that the observation time is large enough when compared to the correlation time of the processes, the covariance matrix of the Fourier coefficient vector $\mathbf{r}(w_l)$ will approach the spectral density matrix

$$\mathbf{K}(w_l) = \mathbf{A}(w_l)\mathbf{P}_s(w_l)\mathbf{A}^H(w_l) + \sigma_n^2\,\mathbf{P}_n(w_l) \qquad (6.5)$$

where $\mathbf{K}(w_l)$, $\mathbf{P}_s(w_l)$ and $\mathbf{P}_n(w_l)$ are the spectral density matrices of the processes $r_i(.)$ $s_k(.)$, $n_i(.)$ respectively. The noise process is assumed to be independent of the sources and the noise spectral density matrix except for a multiplicative constant σ_n^2.

The problem now reduces to the estimation of the τ_i' s from the covariance matrices $\mathbf{K}(w_l)$ and the noise representations. Then the angles of arrival can be computed from the Equation (6.4e).

6.1.2 Problem Solution

This particular approach utilizes a bilinear transformation and dense array approximation to form the transformation matrices. The bilinear transformation matrix that is used can be synthesised from the coefficients of the polynomials $p_k(z) = (1+z)^{M-k}\,(1-z)^{k-1}$, where k =1, 2, ... M-1. M here indicates the number of sensors that the system is using, which in this case is equal to 8. Hence the transformation matrix in this case is an 8x8 matrix, the synthesis of which is shown in the next section.

$\mathbf{E}(w_l)$ denotes a diagonal matrix given by

$$\mathbf{E}(w_l) = \begin{bmatrix} (1+e^{-jw_l\tau_1})^7 & & & \\ & \cdots & & \\ & & \cdots & \\ & & & (1+e^{-jw_d\tau_d})^7 \end{bmatrix} \qquad (6.6)$$

Premultiplying $\mathbf{A}(w_l)$ by the transformation matrix **B** and simplifying the product gives

$$\mathbf{BA}(w_l) = \begin{bmatrix} 1 & \dots & 1 \\ j\tan\frac{w_l r_1}{2} & \dots & j\tan\frac{w_d r_d}{2} \\ \dots & \dots & \dots \\ (j\tan\frac{w_l r_1}{2})^7 & \dots & (j\tan\frac{w_d r_d}{2})^7 \end{bmatrix} \quad \mathrm{E}(w_l) \qquad (6.7)$$

Assuming that the sensor to sensor separation Δ is small when compared to the wavelengths of the incoming signals, $\tan\frac{w_l r_1}{2}$ can be approximated by $\frac{w_l r_1}{2}$.

Now consider an 8x8 diagonal matrix $\mathbf{D}(\frac{w_c}{w_l})$ whose (k,k) th term is given by

$$d_{kk} = (\frac{2w_c}{jw_l})^{k-1} \qquad (6.8)$$

where $w_c = 2\pi f_c$ and f_c is the midband frequency of the signals.

It can be approximated as

$$\mathbf{D}(\frac{w_c}{w_l})\mathbf{BA}(w_l) = \begin{bmatrix} 1 & \dots & 1 \\ w_c r_1 & \dots & w_c r_d \\ \dots & \dots & \dots \\ (w_c r_1)^7 & \dots & (w_c r_d)^7 \end{bmatrix} \quad \mathbf{E}(w_l) \qquad (6.9)$$

There is a new matrix $\mathbf{A}(w_c)$, whose columns are the transformed direction frequency vectors which are dependant upon w_c rather than w_l. The columns of the matrix are linearly independant as long as $r_i \neq r_k$ for $i \neq k$.

A new transformation matrix is defined as

$$\mathbf{T}(\frac{w_c}{w_l}) = \mathbf{D}(\frac{w_c}{w_l})\mathbf{B} \qquad (6.10)$$

This does not depend upon the arrival angles and can hence be computed independently of the angles. Using these transformation matrices for each

individual narrowband frequency, all the spectral estimates can now be combined at the midband frequency in the following manner;

$$\mathbf{G} = \sum_{l=l_1}^{l_1+n_f} \mathbf{T}(w_l)\,\mathbf{K}(w_l)\,\mathbf{T}^H(w_l) \tag{6.11}$$

and $$\mathbf{G_n} = \sum_{l=l_1}^{l_1+n_f} \mathbf{T}(w_l)\,\mathbf{P_n}(w_l)\,\mathbf{T}^H(w_l) \tag{6.12}$$

Then the coherent signal subspace theorem for the matrix pencil $(\mathbf{G},\mathbf{G}_n)$ is used to estimate all the angles of arrivals by computing the maximas of the measure given by

$$J(\theta) = \frac{1}{\sum_{k=d+1}^{8} \| \mathbf{a}_\theta(w_c)\,\mathbf{e_k}(w_c) \|^2} \tag{6.13}$$

where $\mathbf{e_k}(w_c)$ denotes the generalized eigenvectors of the matrix pencil $(\mathbf{G},\mathbf{G}_n)$, which correspond to the 8 - d eigenvalues, and $\mathbf{a}_\theta(w_c)$ represents the new direction frequency matrix.

6.2 PARALLELIZATION AND MODIFICATION

The first objective in the implementation of such signal processing algorithms is to modify them in such a way so that the maximum possible parallelism and pipelining can be achieved which would enable the real time implementation of the algorithm. The modification of the algorithm outlined in the previous section takes into consideration the various tradeoffs involved in the ultimate realization of the hardware like the timing and cost considerations which would make the project viable.

A flow chart of the modified algorithm is shown in Figure 6.1(a). Figure 6.1(b) shows the mathematical transformations that the algorithm involves. There is a linear array of 8 sensors which are sampled at a rate of 80 samples per sec. A segment of 64 samples is considered which form the single step input to the next stage of the FFT processors. As shown in Figure 6.1(a) a single estimation of the angles of arrival involves the processing of 64 such segments of 64 samples each. After 64 samples are collected, the next step involves the transformation of these signals from the time domain to the frequency domain by performing a 64 point FFT. The output is a 33 element vector in the frequency domain which is representative of the input signal at that sensor.

The next block is the calculation of the covariance matrix at each frequency bin. Essentially the covariance matrix consists of the product of the frequency vector and its Hermetian which is obtained from the corresponding elements in the FFT output vectors. Hence for the 33 different narrowband frequencies there are 33 different covariance matrices independant of each other. These matrices are averaged over the 64 segments before being passed on to the next step in the

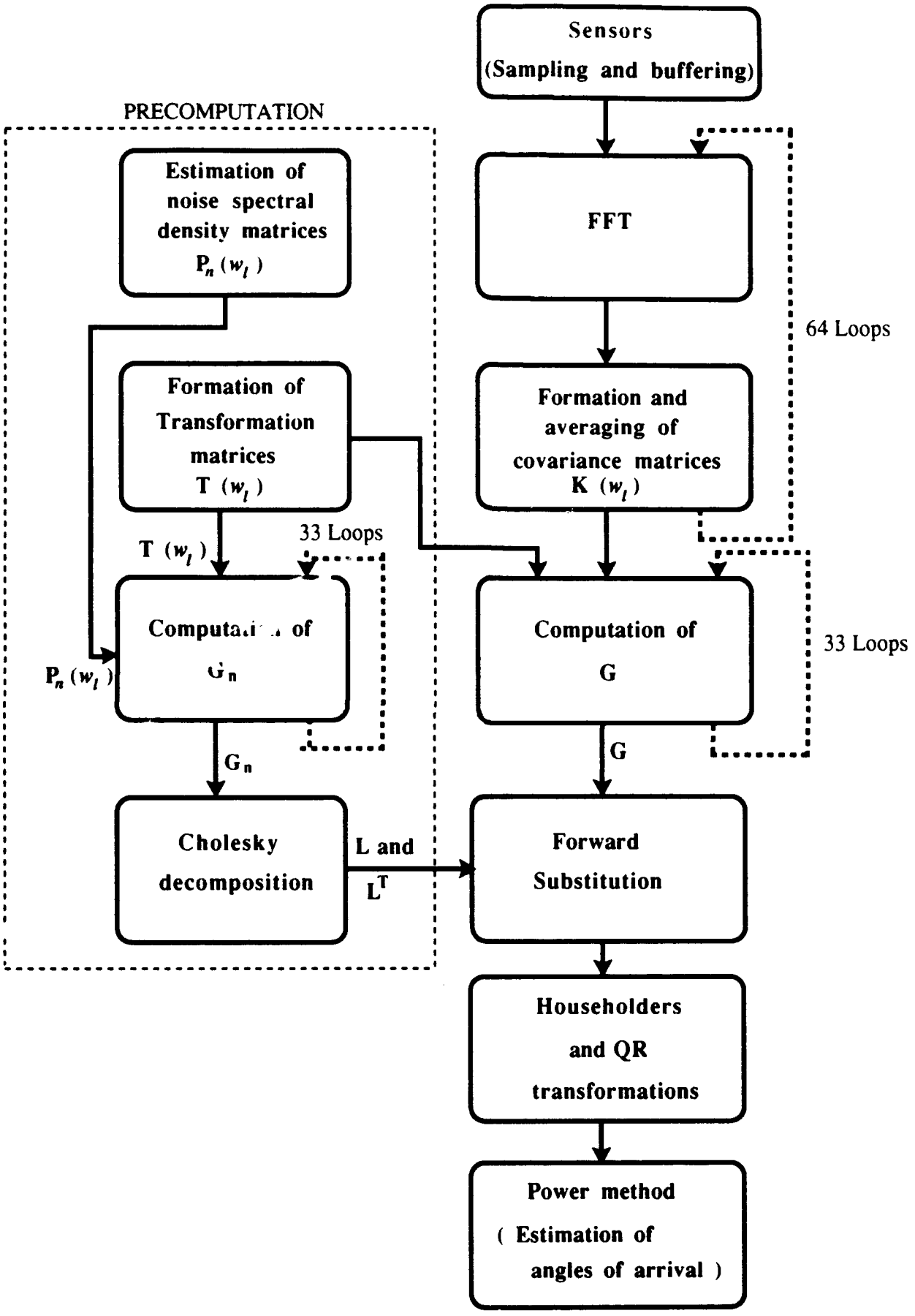
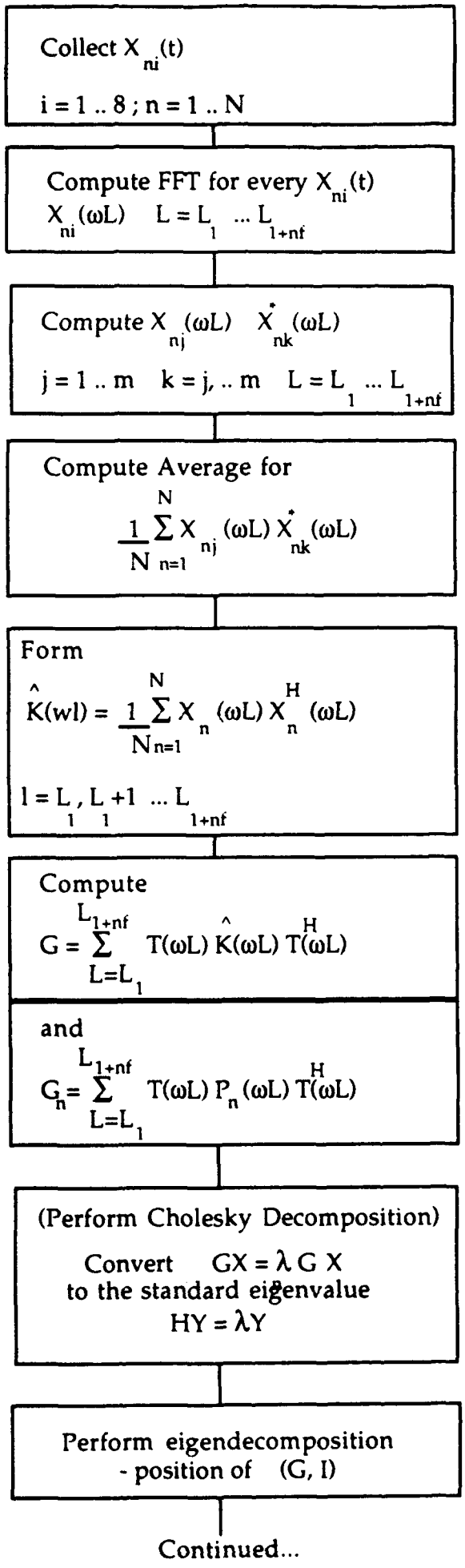


Figure 6.1 : Flowchart of Wideband algorithm


Collect $X_{ni}(t)$
$i = 1 .. 8 ; n = 1 .. N$
Compute FFT for every $X_{ni}(t)$
$X_{ni}(\omega L) \quad L = L_1 \ldots L_{1+nf}$
Compute $X_{nj}(\omega L) \quad X^*_{nk}(\omega L)$
$j = 1 .. m \quad k = j, .. m \quad L = L_1 \ldots L_{1+nf}$
Compute Average for
$\frac{1}{N} \sum_{n=1}^{N} X_{nj}(\omega L) X^*_{nk}(\omega L)$
Form
$\hat{K}(wl) = \frac{1}{N} \sum_{n=1}^{N} X_n(\omega L) X^H_n(\omega L)$
$l = L_1, L_1 + 1 \ldots L_{1+nf}$
Compute
$G = \sum_{L=L_1}^{L_{1+nf}} T(\omega L) \hat{K}(\omega L) T^H(\omega L)$
and
$G_n = \sum_{L=L_1}^{L_{1+nf}} T(\omega L) P_n(\omega L) T^H(\omega L)$
(Perform Cholesky Decomposition)
Convert $GX = \lambda G X$
to the standard eigenvalue
$HY = \lambda Y$
Perform eigendecomposition
- position of (G, I)
Continued...

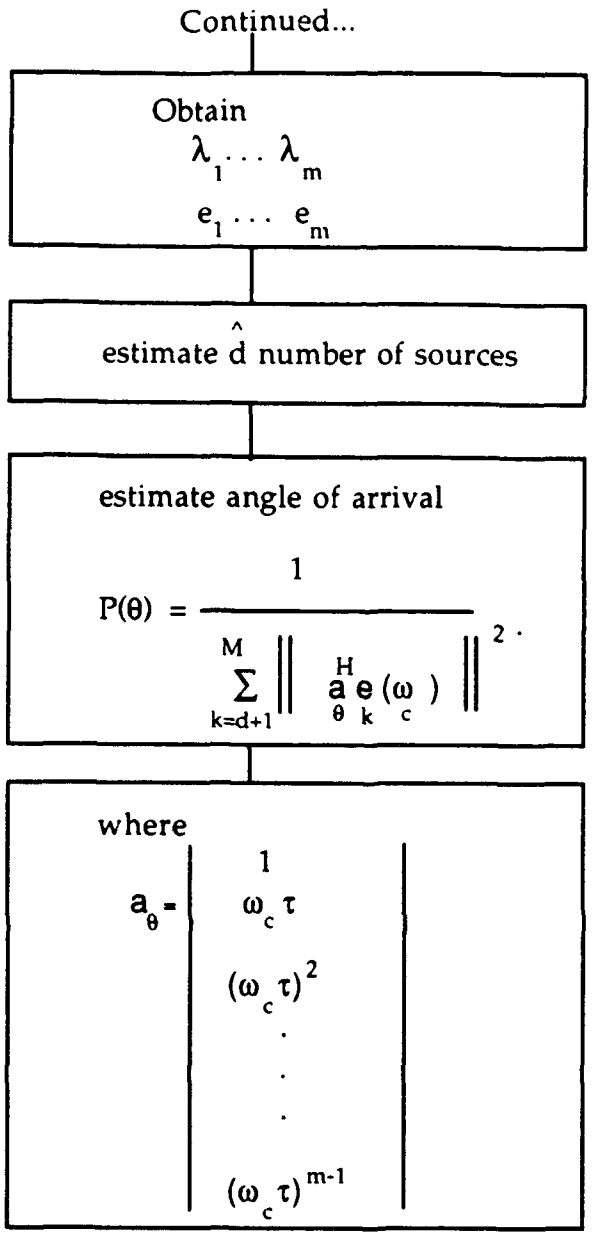


Figure 6.1b Mathematical transformations in the algorithm

algorithm which is the projection of the covariance matrices $\mathbf{K}(w_l)$ onto the single midband frequency in the spectrum to compute the **G** matrix.

The computation of the **G** matrix requires the transformation matrices $\mathbf{T}(w_l)$ which are precomputed as shown in the diagram. As seen from Equation(6.8) in the previous section the computation of the matrix involves the knowledge of the narrowband frequencies in the bandwidth. Given a specific problem such an estimation of the frequency bins is made by splitting the bandwidth into 32 equal parts and taking the frequencies at the boundary. With this initial assumption of the narrowband frequencies in the spectrum of the incoming signals the transformation matrices can be computed offline. This is possible because the matrices are unique for a set of frequencies and are independant of the angles of arrival of the incoming signal. Hence these invariant matrices can be stored in a ROM for a dedicated architecture and can be called up whenever they are required during the processing. However an architectural model has been developed to compute the transformation matrices on line which would enable the system to be more general purpose and allow it to run scans over different frequency ranges without the initial knowledge of their frequency components. The computation of the actual transformation matrices is outlined below following the principles explained in the previous chapter.

6.2.1 Computation of the Transformation matrices

The transformation matrix is derived as follows:

Let **B** be a matrix constructed from the coefficients of the polynomial $p_k(z) = (1+z)^k (1-z)^{8-k}$, where k =1, 2, ... 7. K denotes the number of the row of the

8 x 8 matrix which is formed. In this case the nonsingular matrix has been computed and is shown below

$$\mathbf{B} = \begin{bmatrix} 1 & 7 & 21 & 35 & 35 & 21 & 7 & 1 \\ 1 & 5 & 9 & 5 & -5 & -9 & -5 & -1 \\ 1 & 3 & 1 & -5 & -5 & 1 & 3 & 1 \\ 1 & 1 & -3 & -3 & 3 & 3 & -1 & -1 \\ 1 & -1 & -3 & 3 & 3 & -3 & -1 & 1 \\ 1 & -3 & 1 & 5 & -5 & -1 & 3 & -1 \\ 1 & -5 & 9 & -5 & -5 & 9 & -5 & 1 \\ 1 & -7 & 21 & -35 & 35 & -21 & 7 & -1 \end{bmatrix} \tag{6.14}$$

From this **B** matrix the transformation matrix can be computed according to Equation (6.10). For the matrix $\mathbf{D}(\frac{w_c}{w_l}$, the (k,k) th term is given by

$$d_{kk} = (\frac{2w_c}{jw_l})^{k-1}$$

Let p denote the constant term such that

$$p = \frac{2w_c}{jw_l}$$

The transformation matrix can now be written as shown in Equation (6.15)

The matrix **T** can thus be computed and is stored in a ROM and is retrieved by each processor. The next precomputation block is the calculation of the $\mathbf{G}_n$ matrix which is the estimate of the noise spectral density that is expected to be present in the signal. The algorithm requires a previous knowledge of the noise in the system which is expressed in terms of the P_n matrices at each frequency bin. The procedure for calculating involves two matrix multiplications a; d is similar to the computation of the **G** matrix from the covariance matrices. The calculations are performed 33 times, once for each frequency component and are then averaged at the midband frequency.

$$\mathbf{T}(w_l) = \begin{bmatrix} p & 7p^2 & 21p^3 & 35p^4 & 35p^5 & 21p^6 & 7p^7 & p^8 \\ p & 5p^2 & 9p^3 & 5p^4 & -5p^5 & -9p^6 & -5p^7 & -p^8 \\ p & 3p^2 & -p^3 & -5p^4 & -5p^5 & -p^6 & 3p^7 & p^8 \\ p & p^2 & -3p^3 & -3p^4 & 3p^5 & 3p^6 & -p^7 & -p^8 \\ p & -p^2 & -3p^3 & 3p^4 & 3p^5 & -3p^6 & -p^7 & p^8 \\ p & -3p^2 & p^3 & 5p^4 & -5p^5 & -p^6 & 3p^7 & -p^8 \\ p & -5p^2 & 9p^3 & -5p^4 & -5p^5 & 9p^6 & -5p^7 & p^8 \\ p & -7p^2 & 21p^3 & -35p^4 & 35p^5 & -21p^6 & 7p^7 & -p^8 \end{bmatrix} \quad (6.15)$$

The equation governing this transformation is shown below.

$$\mathbf{G_n} = \sum_{l=l_1}^{l_1+n_f} \mathbf{T}(w_l)\, \mathbf{P_n}(w_l)\, \mathbf{T}^H(w_l) \quad (6.16)$$

The matrix $\mathbf{G}_n$ is then stored in the ROM and accessed at the time of the Cholesky decomposition.

6.2.2 Computation of G

The G matrix which is the combination at the midband frequency of all the individual covariance matrices of different narrowband components requires the projection of these matrices by the transformation matrices and involves two matrix multiplications as shown in the equation below.

$$\mathbf{G} = \sum_{l=l_1}^{l_1+n_f} \mathbf{T}(w_l)\, \mathbf{K}(w_l)\, \mathbf{T}^H(w_l) \quad (6.17)$$

The process goes through 33 iterations as shown in the flowchart. Each loop involves two matrix multiplications which are done sequentially, because the input to the second operation is the output from the first. However parallelism has been achieved inside each operation as it is performed in one cycle. The computation of the **G** matrix gives the matrix pencil $(\mathbf{G}, \mathbf{G}_n)$ of which $\mathbf{G}_n$ has been precomputed.

6. 2.3 Cholesky decomposition

The further processing of the signal requires that it be organised into a standard form so that certain standard operations of matrix algebra like the eigendecomposition can be performed. The algebraic manipulations which are performed to achieve the objective are described below.

$\mathbf{G}_n$ and **G** are two matrices which need to be put in the standard form such that

$$\mathbf{G}\,\mathbf{X} = \lambda\,\mathbf{G}_n\,\mathbf{X} \tag{6.18}$$

where λ = the eigenvalues of **G**

X = the eigenvector matrix of $\mathbf{G}_n$ and **G**

Decomposing $\mathbf{G}_n$ into

$$\mathbf{G}_n = \mathbf{L}\,\mathbf{L}^T \tag{6.19}$$

and substituting $\mathbf{G}_n$ in the equation and multiplying both sides by $\mathbf{L}^{-1}$ gives

$$\mathbf{L}^{-1}\,\mathbf{G}\,\mathbf{L}^{-T}.\,\mathbf{L}^T\,\mathbf{X} = \lambda\;\mathbf{L}^{-1}\,\mathbf{L}\,\mathbf{L}^T\,\mathbf{X}$$

Defining $\mathbf{L}^{-1}\,\mathbf{G}\,\mathbf{L}^{-T} = \mathbf{H}$ and $\mathbf{L}^T\,\mathbf{X} = \mathbf{Y}$ (6.20)

The standard form required for eigendecomposition can be written as

$$\mathbf{H}\,\mathbf{Y} = \lambda \mathbf{Y} \tag{6.21}$$

The decomposition $\mathbf{G}_{n} = \mathbf{L}\,\mathbf{L}^{T}$ is obtained by doing the Cholesky decomposition which is the next step in the algorithm as shown in the flowchart.

The flowchart of the Cholesky decomposition is shown in Figure 6.2. The objective of reducing to a lower triangular matrix is achieved by computing the elements below the diagonal according to the equation

$$a_{ki} = \frac{a_{ki} - \sum_{j=1}^{i-1} a_{ij}a_{kj}}{a_{ii}} \tag{6.22}$$

The diagonal elements are however computed by the formula

$$a_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} a_{kj}^{2}} \tag{6.23}$$

Once the lower triangular matrix **L** has been computed the transpose $\mathbf{L}^{T}$ can be obtained.. The next step is to obtain the two matrices **H** and **Y**. This part needs the calculation of the inverse of the lower triangular matrix **L** as is seen from Equation(6.20). This computation is both time consuming and complex especially for real time applications. The ultimate objective is not to calculate the inverse and to circumvent this requirement, a simple algebraic manipulation is described below:

Assuming a matrix **W** such that

$$\mathbf{L}\,\mathbf{W} = \mathbf{G} \tag{6.24}$$

we have

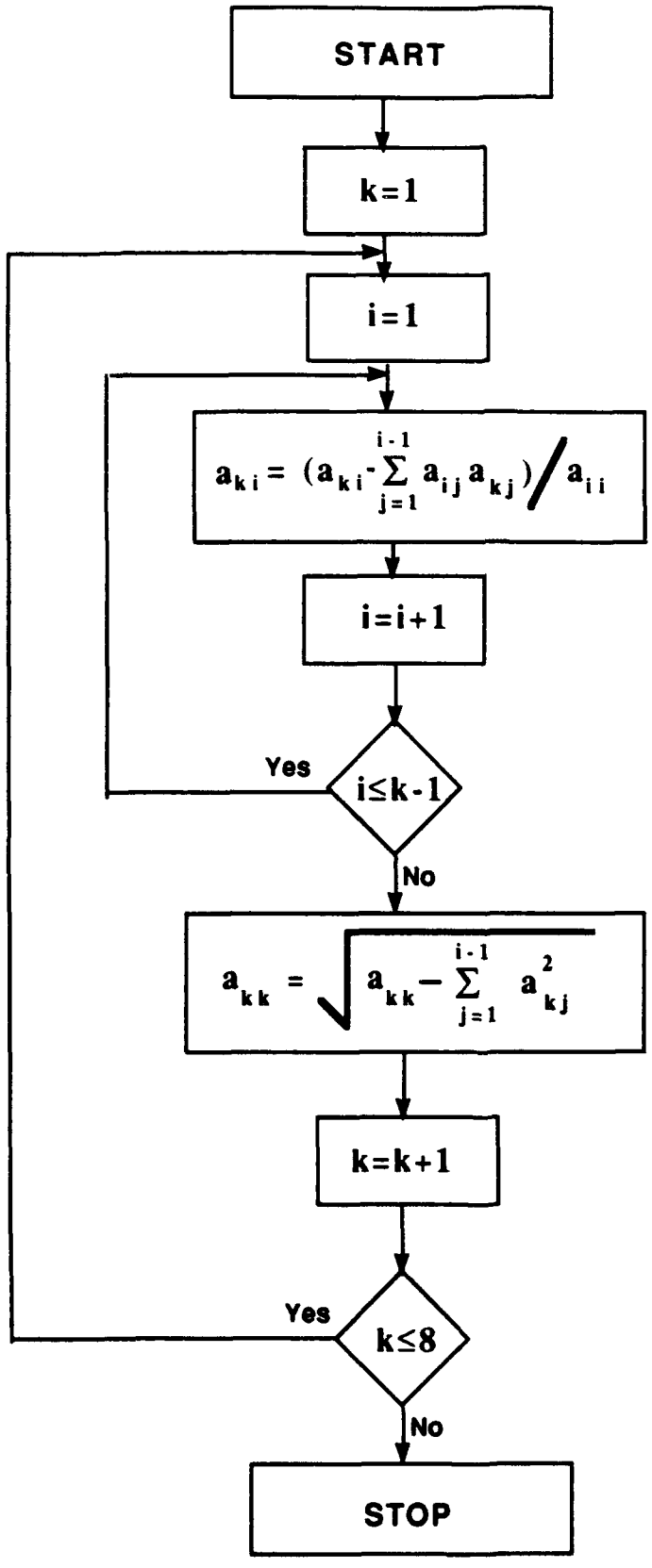


Figure 6.2- Flowchart for Cholesky Decomposition

$$W = L^{-1} G$$

Taking the transpose and premultiplying both sides of Equation (6.24) by L^{-1} gives

$$L^{-1} W = L^{-1} (L^{-1} G)^T$$

$$= L^{-1} G^T (L^{-1})^T$$

$$= L^{-1} G (L^{-1})^T \qquad \text{(as } G \text{ is Hermitian)}$$

$$= H$$

Hence

$$L H = W^T \tag{6.25}$$

Considering the two Equations (6.24) and (6.25) it can be seen that the problem of computing the inverse is now reduced to the computation of the **H** matrix by two forward substitution operations. First the matrix **W** is computed from the Equation (6.24) as the other two matrices are known. Then it is transposed, which is a simple routing exercise in the architecture and use the result in Equation (6.25) to compute the **H** matrix.The computation of **Y** also follows the same procedure. The resultant matrices can now be treated in the same manner as the narrowband case to compute the angles of arrival. First the Householders and QR transformations are performed to reduce the dense matrix into a diagonal one and then the power method is used to compute the angles of arrival. The description of these methods is given in the previous chapters dealing with the narrowband case.

6.3 HARDWARE IMPLEMENTATION

In the hardware implementation of the proposed algorithm it is necessary to consider the tradeoffs between the timing requirements and the number of processors in each stage. Though parallelization and pipelining of most tasks in the process is possible this would require a large number of processing elements which are not really necessary as far as the timing requirements are concerned because the processing speed is going to be determined by the sampling rate at the sensors which is not very high.

The overall block diagram of the architecture is shown in Figure 6.3. The first part shows the sensors and the buffering stage. The input to the FFT processors is a 64 element vector. Hence a buffering stage is provided to store and accumulate one segment of 64 samples. The buffer has a control mechanism to coordinate data flow from the FFT processors. The data is transferred to all the processors simultaneously a sample at a time.

The next stage is that of the FFT processors. In this algorithm the computation of the angles of arrival is done in the frequency domain so the first operation that is performed on the incoming data is the Fourier transform. The DSP 56000 chip is used to calculate the FFT for the data from each sensor. From the specifications of the chip it has been calculated that it can perform the 64 point FFT in about 120 μs, which is acceptable for this algorithm. The output from the FFT processors is a 64 element vector in the frequency domain. But the components of the vector are symmetrical and hence for computation purposes only one side of the spectral elements is considered. This reduces to a vector of 33 elements which is used to compute the covariance matrices. The architecture developed for the covariance matrix attempts to keep the symmetry of using 8

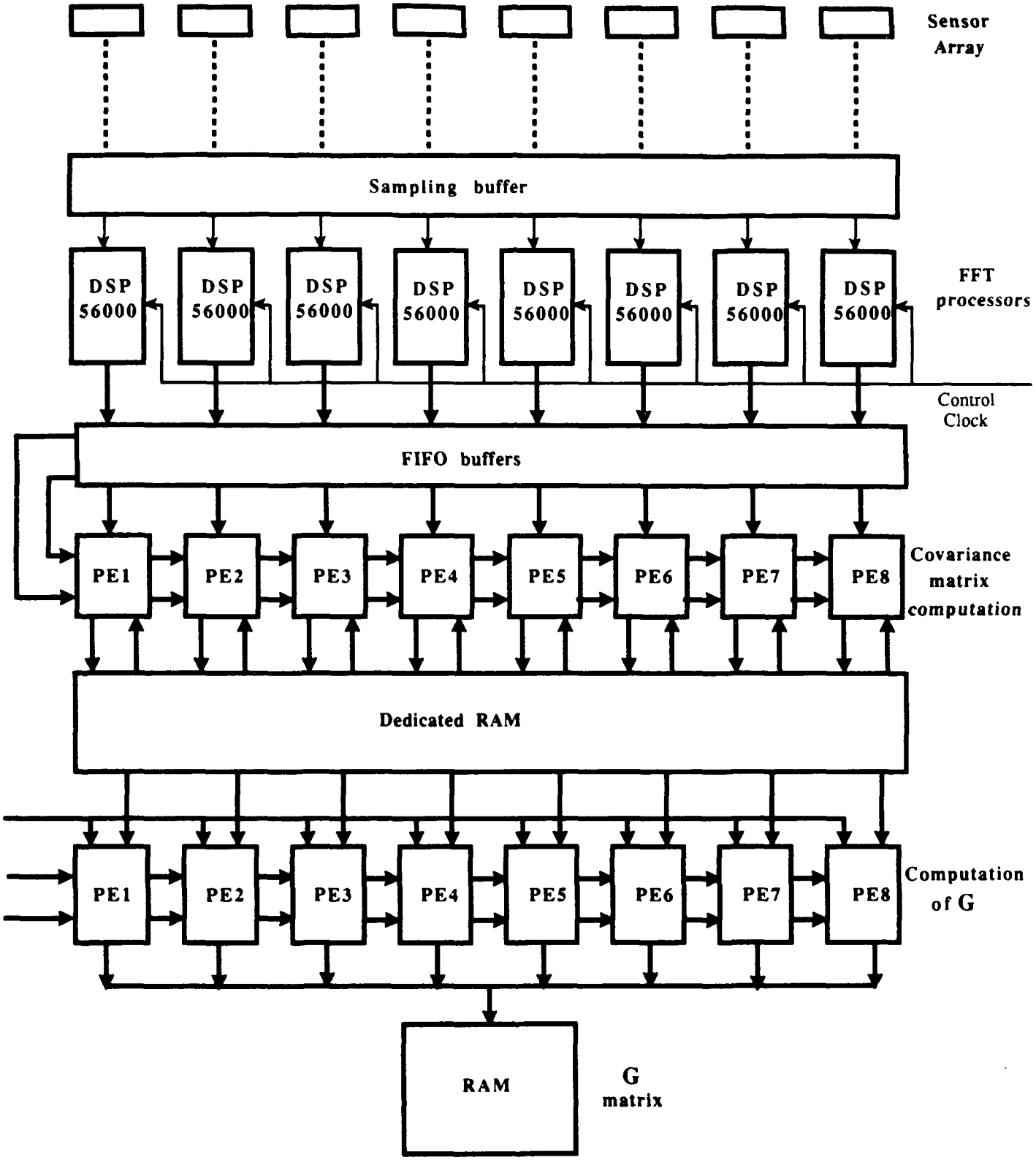
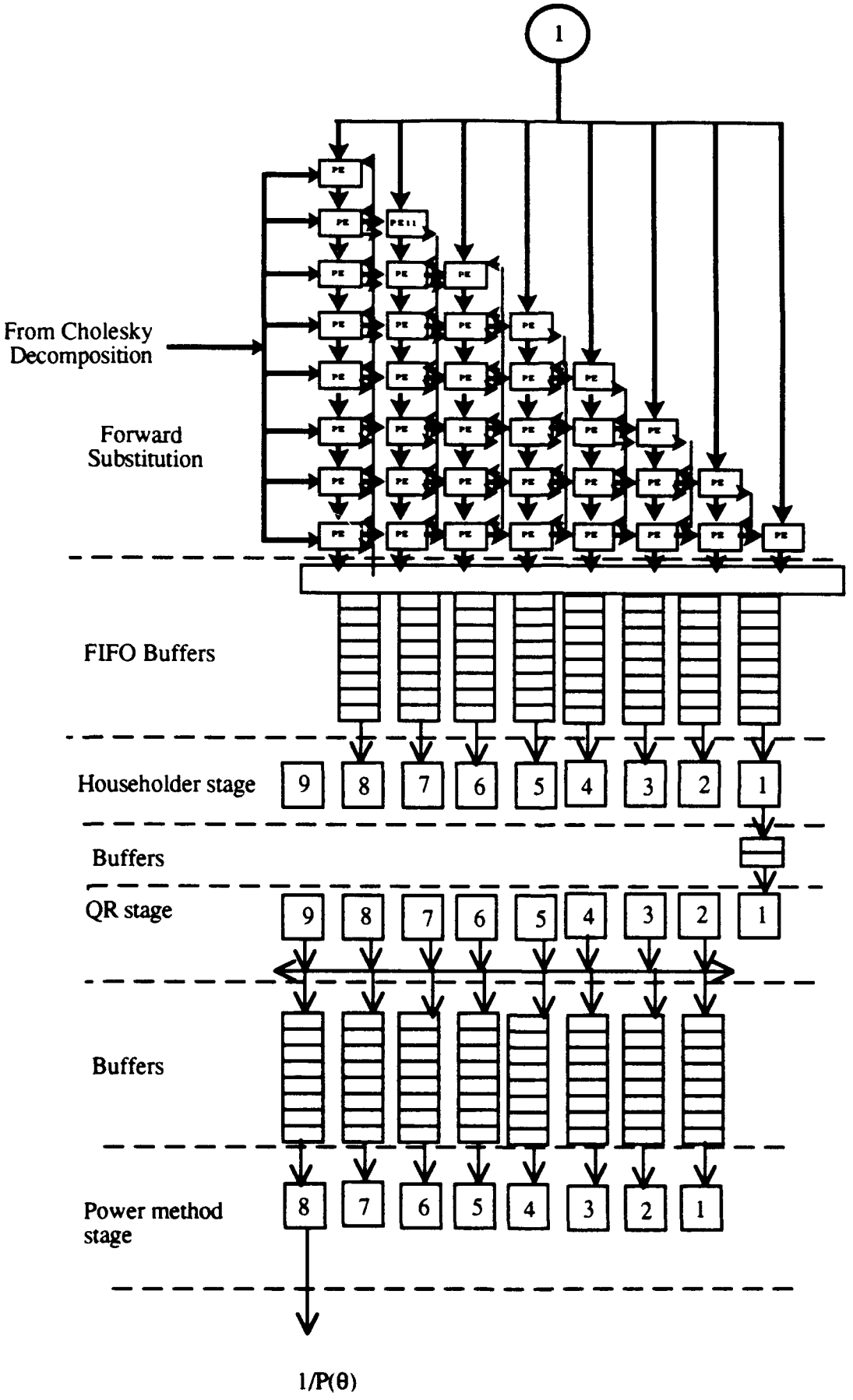


Figure 3- Overall system architecture till computation of G


1
From Cholesky Decomposition
Forward Substitution
FIFO Buffers
Householder stage
9 8 7 6 5 4 3 2 1
Buffers
QR stage
9 8 7 6 5 4 3 2 1
Buffers
Power method stage
8 7 6 5 4 3 2 1
1/P(θ)

processors for each stage. A set of FIFO buffers is used between each set of processors to store the results from the FFT operation. A clock signal as shown in the figure is used to retrieve the data from the buffers in a synchronous mode which is necessary for the input to the covariance matrix processors.

6.3.1 Covariance matrix computation

The computation of the covariance matrix at each frequency bin essentially involves the multiplication of two 8 element vectors. These correspond to the frequency component at each of the sensors and are indicative of the change in the observed signal between the sensors. Figure 6.4 shows a more detailed diagram of the architecture for the computation of the covariance matrix. As shown in Figure 6.4 this stage consists of 8 processors each of which is used to compute one column of the covariance matrix. The flowchart in Figure 6.5 shows the various steps involved in the calculation of the 33 matrices. Figure 6.6 shows a more detailed diagram of the processors in the covariance stage.

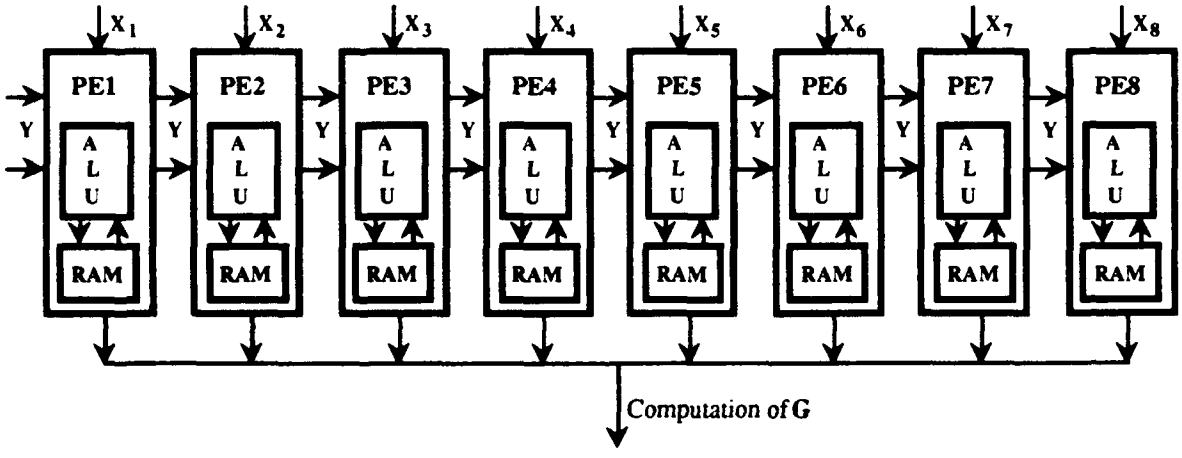


Figure 6.4 - Architecture for computation of covariance matrices

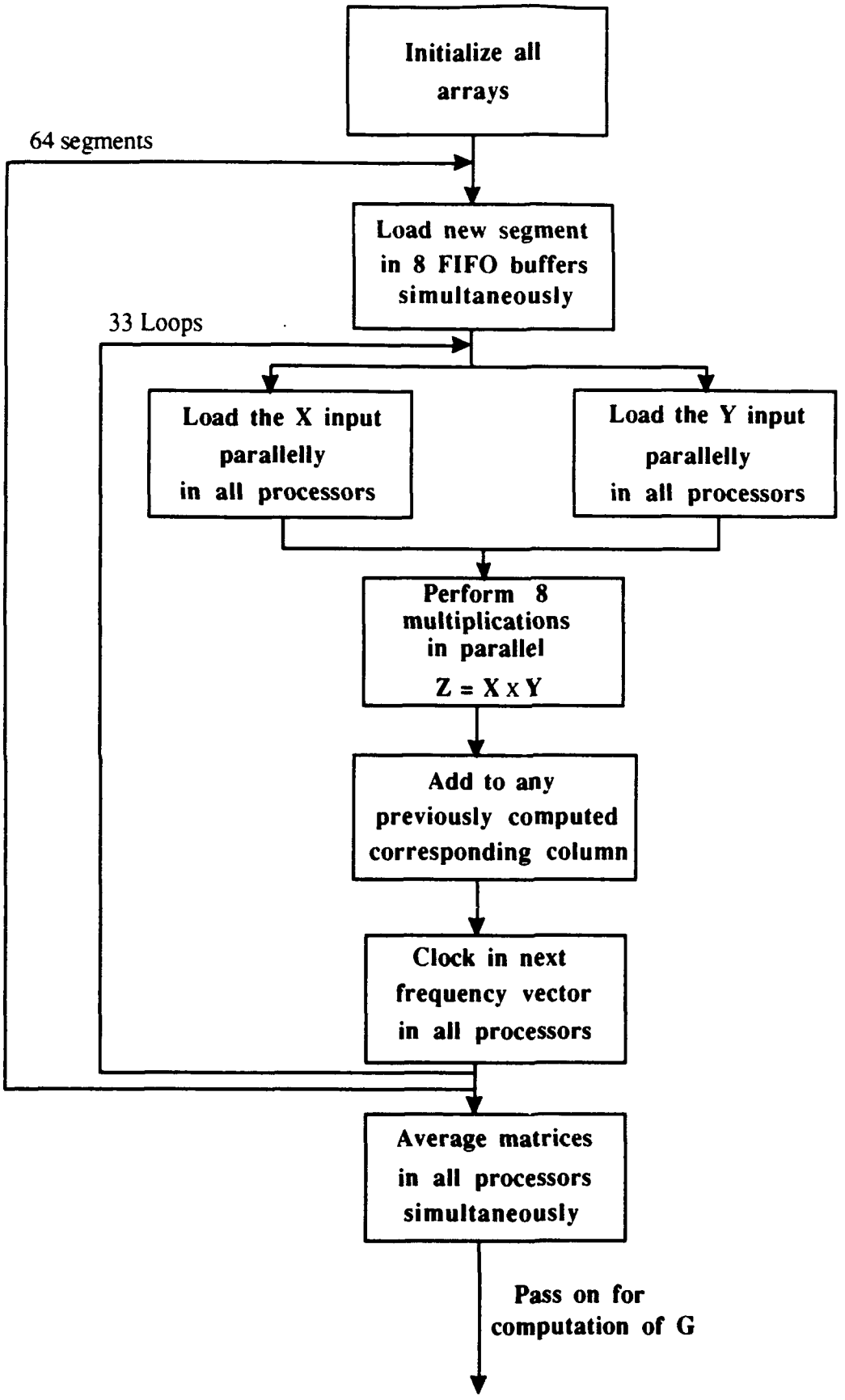


Figure 6.5 - Flowchart for computation of covariance matrices

Note: This flowchart is executed by all procesors in parallel

Basically the computation of the covariance matrix involves the multiplication of a vector with its transpose resulting in a square matrix whose

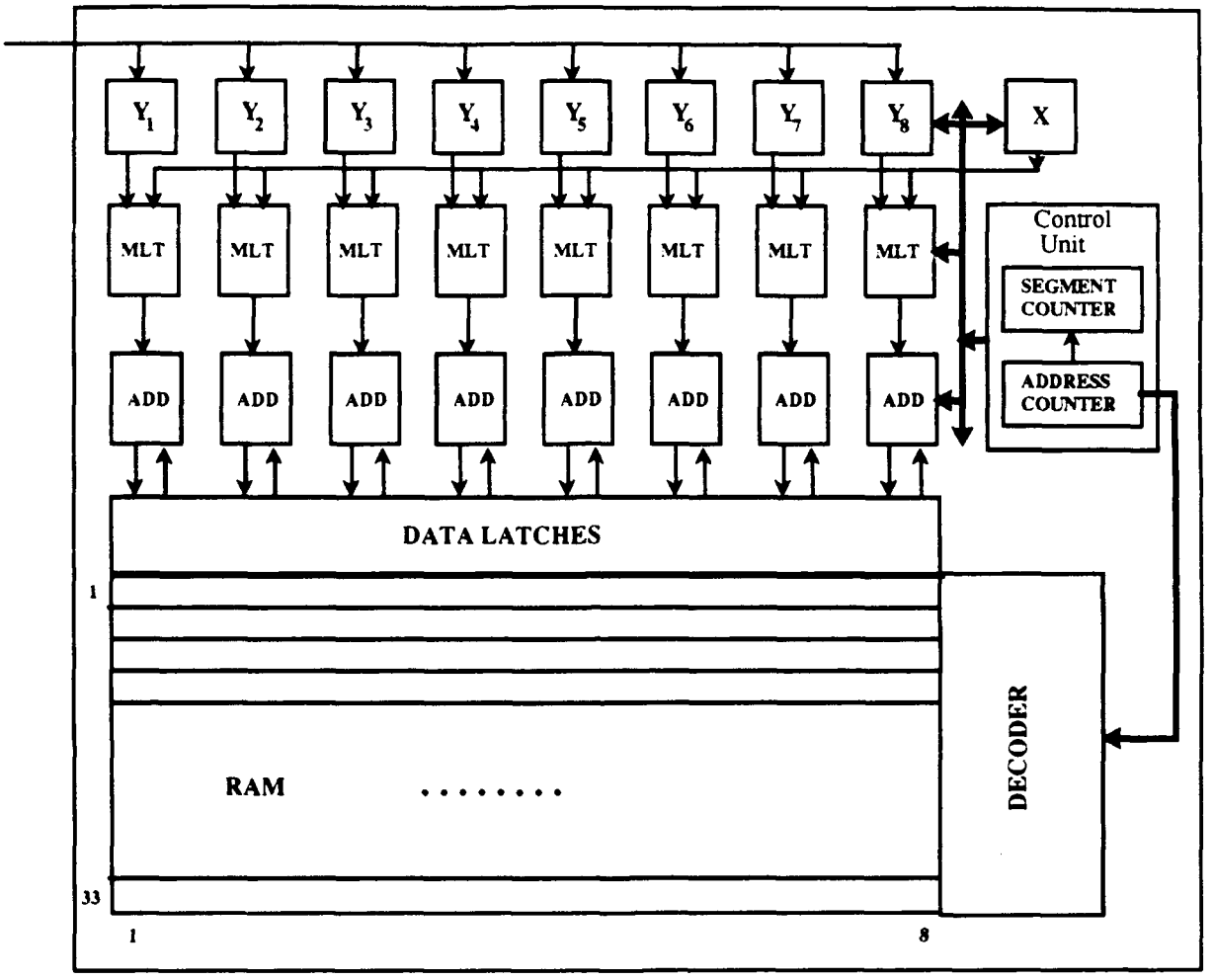


Figure 6.6 Processing Element for covariance matrix stage

dimensions are the size of the vector. In this case the number of elements in the vector is 8, which gives a 8x8 covariance matrix. This also permits the mapping of the computation process in an array of 8 processors, each of which calculates one column of the resultant matrix. Each column is formed by the product of that particular element with the whole vector. For example the third column (which is computed by the third PE), is formed by the product of the third element with the entire column. Hence the inputs to the third processor will be the third

element (**X**) and the vector (Y_1 - Y_8). These elements are obtained from the FIFO buffers in which the output from the FFT processors are stored. The loading of these elements can be achieved in parallel with a multiplexed bus which will route the data from a buffer to a single PE (**X** input) and a broadcast bus which will put the data to all the processors (Y_1 - Y_8 inputs). As seen from Figure 6.6, once the data is latched in to the buffers inside the PE, it is passed on to a complex multiplier. The two multiplier inputs are the **X** value and the corresponding **Y** value. Once the product is computed, it is passed on to an adder, which adds the incoming value to one that has been computed from the previous segment. This previous value is stored in a local RAM as shown in Figure 6.6 and can be retrieved as follows.

The control unit inside the PE basically has the function of supplying the various signals which would enable the correct data to be retrieved from the local RAM during the arithmetic operations. An address counter which runs from 1 to 33 will generate the address which is needed to retrieve the proper vector from the RAM. The decoder takes the signal from the counter and enables a particular row which contains the vector corresponding to that frequency. The particular vector is put on the data latches from where it goes to the adder. This completes the read cycle from the memory. Once the addition is done, the data is now written back into the latch overwriting the data which had been previously stored. A write cycle is executed and the acccumulated result is written back into the same memory cells. The address is held valid till the write operation is completed. The counter is now incremented which takes the whole operation into the next cycle. Once the counter completes 33 cycles it is reset and a pulse is sent to the segment counter which is incremented. The segment counter is set to run from 1 to 64 and is used to indicate the end of a frame.

The memory is organized into an array of 8 x 33 cells. Each cell is capable of storing one element of the vector. The word length is such that 8 elements can be accessed in one cycle on parallel data buses. The addresses run from 0 - 32 for the 33 vectors that are stored. Once the computations have been performed for one frame they are averaged, and passed on for the computation of **G**.

6.3.2 Computation of G

The computation of the **G** matrix reduces the 33 frequency matrices into one single matrix. An important aspect to note is that this computation is required to be done only once every frame, i.e. every 64 segments. The architecture is very similar to the one used for the covariance matrix computation except that the operations are now matrix based instead of being vector based. This calls for a slight change in the memory requirements and the operations in the computation. As shown in Figure 6.3 there is an array of 8 processors each one of which is used to compute one column of the resultant matrix.

The formation of the **G** matrix involves two matrix multiplications, which are used to project the 33 frequency matrices into a single combined matrix at the central frequency according to the equation below.

$$\mathbf{G} = \mathbf{T}(w_l)\,\mathbf{K}(w_l)\,\mathbf{T}^{\mathbf{H}}(w_l)$$

As the matrices are 8x8, the operations are mapped in an 8 processor array as shown in Figure 6.3. Each processor computes one column of the resultant matrix. The data routing is a bit more complex this time because the operands are matrices which need to be loaded into each processor. To simplify this problem the architecture is configured in such a way that only one column needs to be unique to each processsor. In this case it would be the *i*th column of

the $\mathbf{T}^{\mathbf{H}}(w_l)$ matrix going to the *i*th processing element. The rest of the data (i.e. the $\mathbf{T}(w_l)$ and the $\mathbf{K}(w_l)$ matrices are broadcast simultaneously to all the processors during the computation. The $\mathbf{T}(w_l)$ and the $\mathbf{T}^{\mathbf{H}}(w_l)$ matrices can be precomputed, as they are independant of the angles of arrival and are dependant only on the frequency spectrum, which is known a priori. Hence they can be stored in an external ROM and retrieved whenever required. The computation of a column of **G** at each processor can be done by two consecutive multiplications of an 8x8 matrix with an 8x1 vector each of which results in an 8x1 column vector. The first operation is multiplying the covariance matrix $\mathbf{K}(w_l)$ to the *i*th column of the $\mathbf{T}^{\mathbf{H}}(w_l)$ matrix, which gives the *i*th column of the $\mathbf{K}(w_l)\,\mathbf{T}^{\mathbf{H}}(w_l)$ matrix. Next the $\mathbf{T}(w_l)$ matrix is multiplied to the previous result which gives the *i*th column of the **G** matrix at the *i*th processor.

A flowchart of the process of computation of the **G** matrix is shown in Figure 6.7. The algorithm has been parallelized so that the processor can execute nonsequential operations at the same time. The first operation is the loading of the two input vectors, which are done simultaneously. The next set of operations involve the parallel multiplication of the vector elements. At the same time the next row of the $\mathbf{K}(w_l)$ matrix can be loaded into the input latch. Also from the second loop onwards the results can be accumulated. Next the eight elements are added to give the innerproduct which is one element of the column. This repeats for eight loops to compute all the elements of the 8x1 column.

Similarly the second matrix multiplication is performed except that this time the **X** input is the resulting column of the first multiplication and the **Y** input is the row of the $\mathbf{T}(w_l)$ matrix. This operation is repeated eight times to compute the **G** matrix for the first frequency bin. The process then runs through 33

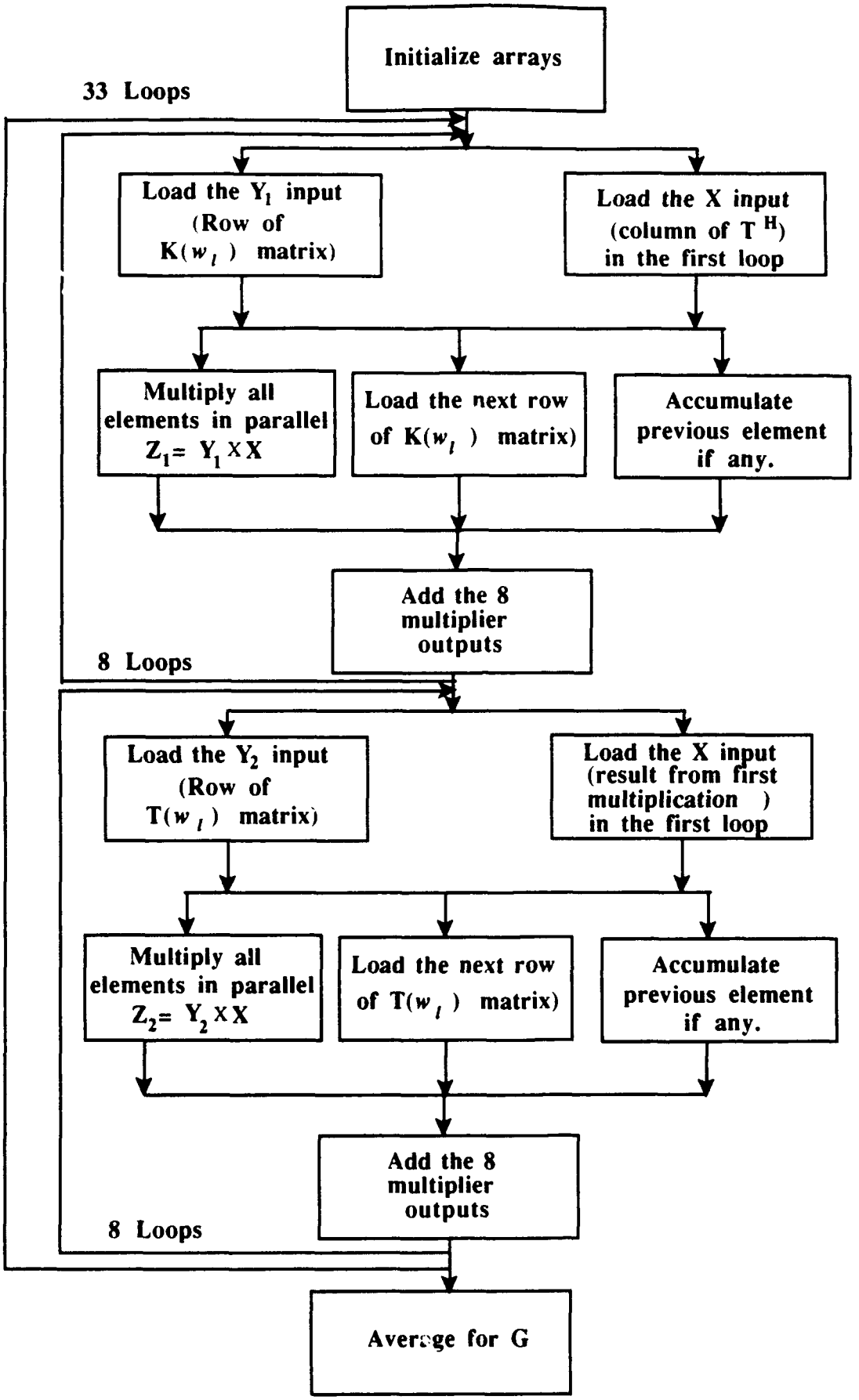


Figure 6.7 Flowchart for the computation of G matrix

iterations for the 33 frequencies. The values are averaged and the final **G** matrix is calculated.

The internal block diagram of the PE used for the calculation of the **G** matrix is shown in Figure 6.8. The **X** input is the *i*th column vector of $\mathbf{T}^{\mathbf{H}}(w_l)$. For the fourth processor the input would consist of the fourth column of the $\mathbf{T}^{\mathbf{H}}(w_l)$ matrix. The loading can be done in parallel, to all the processors. the

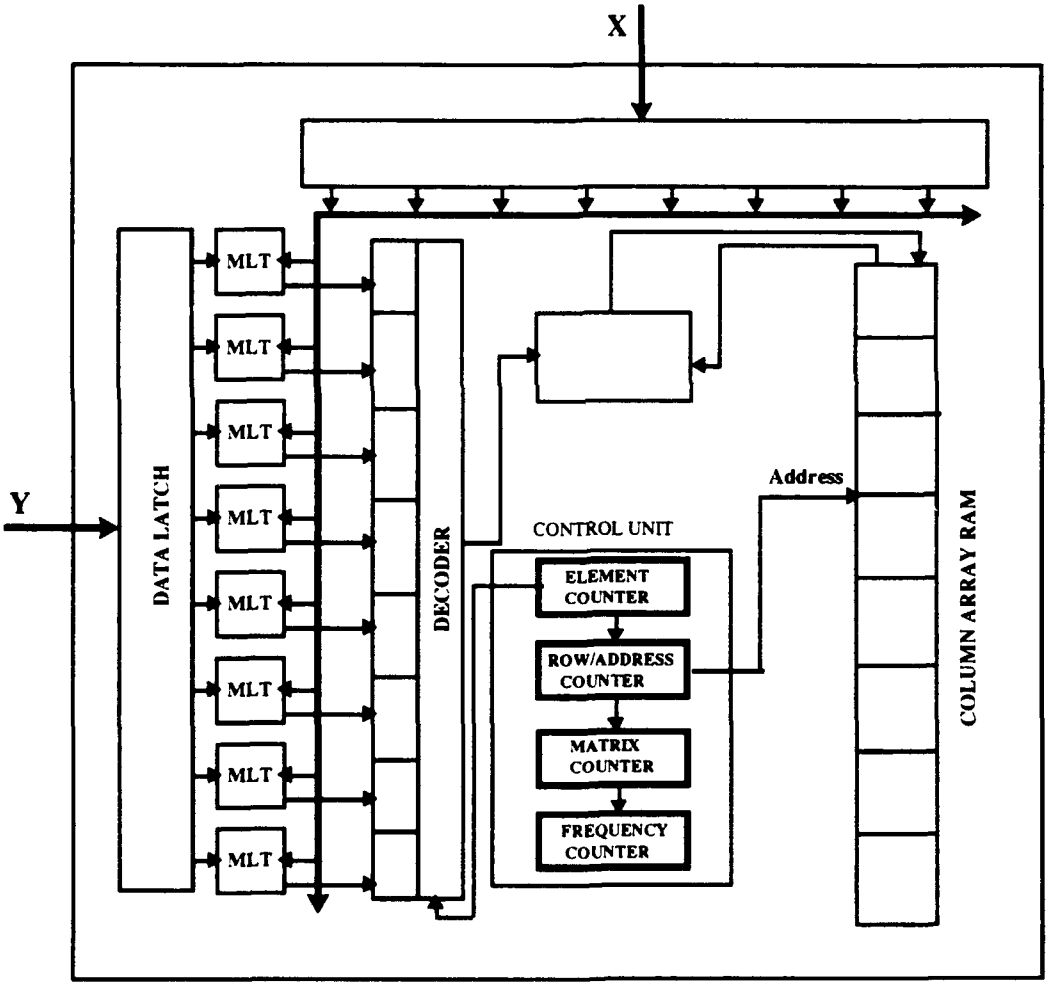


Figure 6.8 - Processing element for computation of **G**

other input consists of the $\mathbf{K}(w_l)$ and the $\mathbf{T}(w_l)$ matrices. The sequence of operations is shown in the flowchart and has been explained above. The eight

multipliers perform the eight complex multiplications required to form the innerproduct in parallel. The results are fed through a multiplexer to an adder which sums them up, and stores the result in the memory array which can be retrieved for later processing. The new row for the next loop is loaded into the data latch when the multiplications are being performed. Once the process goes into the second frequency the adder also has to retrieve the data from the array and add to the newly computed value. This operation is performed by first reading the data from the RAM, adding it and writing the result back into the same memory location.

The control unit essentially consists of four counters which are used to keep track of various operations being performed. The first counter is the element counter which upcounts to eight and is used to control the innerproduct computation. It enables the latches, which load the data from the appropriate mulltiplier in to the adder. Once the element counter counts eight, it is reset and a pulse is sent to the row/address counter which is incremented. The row/address counter also counts to eight and keeps track of the row of the input matrix that is being loaded. This counter also provides the address for the RAM to store and retrieve the data. The third counter is a matrix counter which counts the matrix multiplications. It is a simple inverter and specifies the first or second multiplication. This is complemented every time the row/address counter is reset. The output of the matrix control is used to load the appropriate matrix into the processor. The last counter is the frequency counter which counts upto thirty three frequency bins. The outputs from the last two counters are basically used to retrieve the appropriate data from the buffers. Once the G values are computed for all the frequency bins, the processor then averages the column to give the

value of the column of G. The whole matrix is obtained from the columns from the eight processors.

6.3.3 Computation of G_n

This DOA algorithm requires the knowledge of the noise spectra in the signal which is finally expresssed in the form of the G_n matrix. The G_n matrix can be computed similar to the G matrix except that the signal vectors are replaced by sampled signals which do not have any wavefronts from the objects in them. i.e. they are representative of the medium only. This operation needs to be performed only for updating the G_n matrix. As explained in the previous section there is one operation which is performed on the G_n matrix which is not performed on the G matrix, which is the Cholesky Decomposition. This operation is required to put the two matrices into the standard form for further processing. The Cholesky decomposition can be carried out effectively offline from the main processing stream, and the result fed back online whenever the need arises. The architecture for this operation is explained in Section 6.3.5.

6.3.4 Forward Substitution

As explained in the previous section the G matrix needs to be decomposed into a standard form. This transformation is accomplished by performing two forward substitution operations as explained in Section 6.2.3. The steps in the forward substitution are more complex than the previous stages because the operation involves a series of multiply and accumulate steps to calculate each element. Hence to reduce the complexity of the PEs, a systolic architecture is adopted for this stage. Figure 6.9 shows a completely parallel and pipelined proposed architecture for this operation.

The stage consists of an array of 8x8 processors each of which computes one element in the matrix. A detailed figure of a typical processing

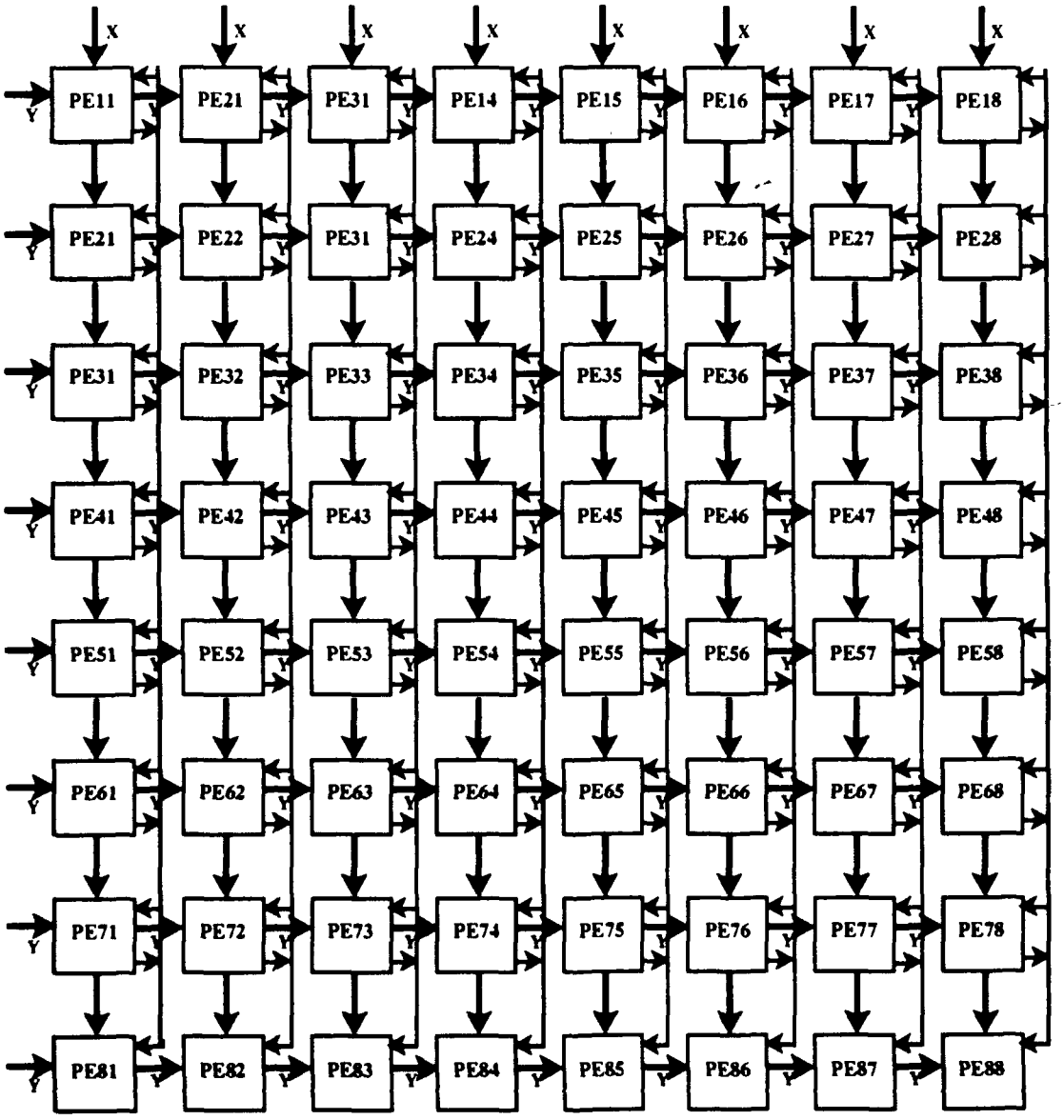


Figure 6.9 - Fully pipelined and parallel architecture for the Forward Substitution operation

cell is shown in Figure 6.10. The Y input in this case is the particular column of the lower triangular matrix L and the X input is the corresponding element from the G matrix. As before the X input is unique to the PE while the Y input is

broadcast to all the PEs in that column. All the outputs are transmitted downwards for further processing. In the first cycle the first row elements are

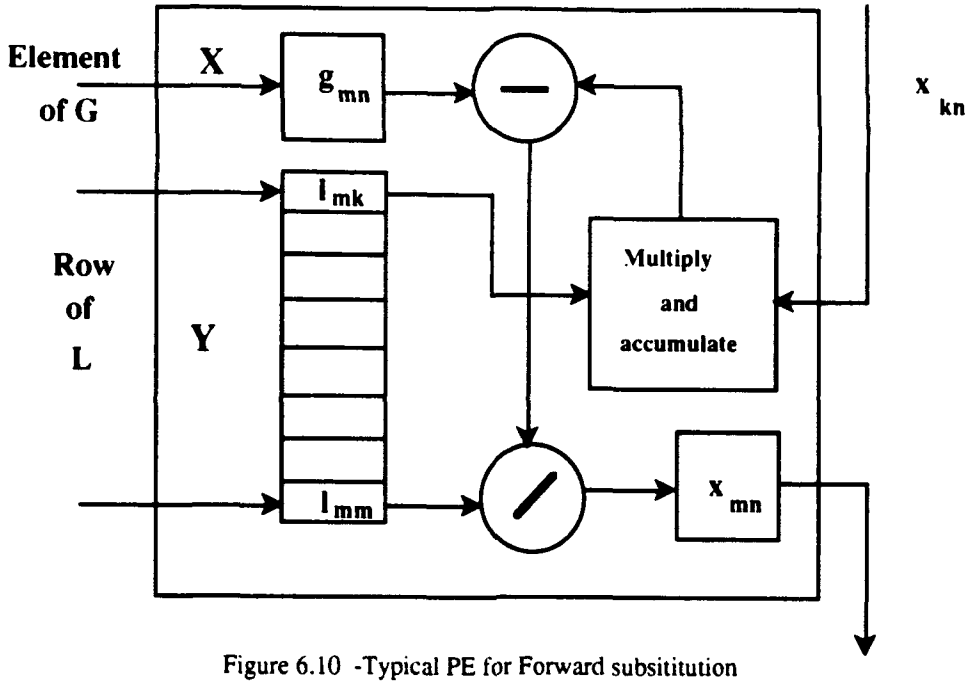


Figure 6.10 -Typical PE for Forward subsititution

computed. The result is broadcast to all the processors directly beneath it. From the second cycle onwards the processsors beneath the row of that particular operation, will be active while those which have already calculated their corresonding elements are inactive. The whole process of calculation of the result takes eight cycles. After it is done, the next set of data is loaded to compute **H** for the standardization.

6.3.5 Cholesky Decomposition

The flowchart for the Cholesky decomposition shows the various sequence of steps which the processors have to perform. Figure 6.11 shows the array which is used for Cholesky decomposition of the G_n matrix. The triangular array is loaded into the processors with each element going to its corresponding

processor. The processors along the diagonal are different from the processors below it as they have different computations to perform. The computation

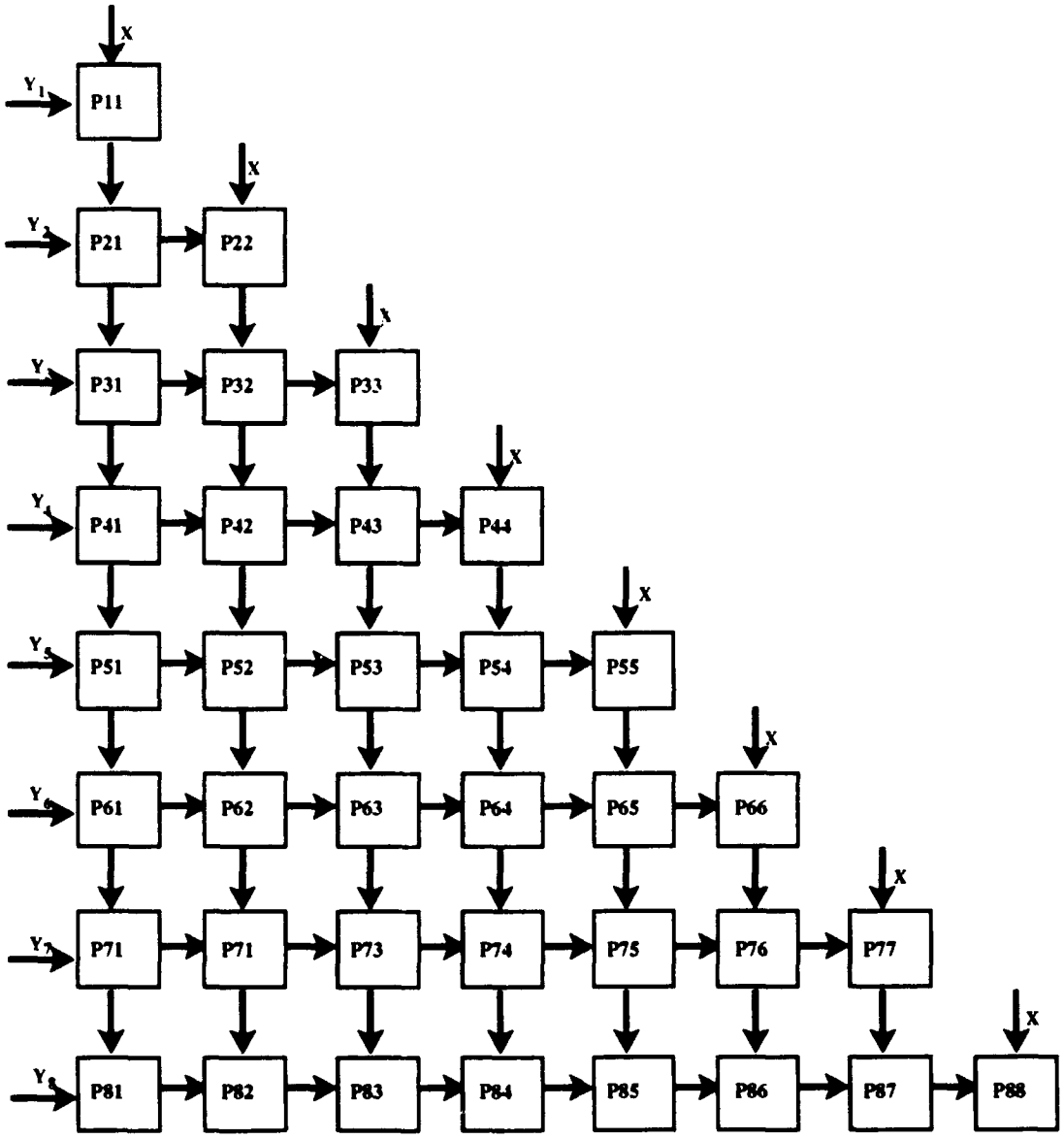


Figure 6.11- Architecture for Cholesky Decomposition

process takes eight cycles during each of which one column of the resultant L matrix is computed.

The initial inputs are the individual elements of the matrix. Unlike the previous processes, the input to the processors in the Cholesky decomposition change during every cycle depending upon the number of the column that is

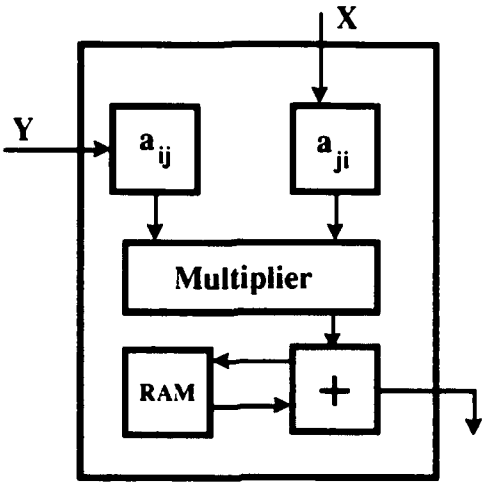


Figure 12 - Processing element for Cholesky Decomposition

being computed.The **X** input to a PE will be the above diagonal elements of the corresponding column while the **Y** inputs are the corresponding elements from the same row. The results are accumulated after every multiplication. For example when the sixth row is being computed, there will be five multiplications and additions before the final subtraction and division. The accumulated value is subtracted from the original element value and then divided by the column's diagonal element. The equation for computing the subdiagonal element is

$$a_{ki} = \frac{a_{ki} - \sum_{j=1}^{i-1} a_{ij}a_{kj}}{a_{ii}}$$

and the diagonal elements are computed by the equation

$$a_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} a_{kj}^2}$$

Hence the PEs on the diagonal have a slightly different function to perform than the PEs below the diagonal and hence are a little different.

Once the 33 spectral matrices have been combined at a single frequency then the computation can be carried out by the Householder/QR transformations and the power method. The developed architectures for these methods have been explained in the previous chapters on the narrowband case.

6.3.5 Conclusions

An architecture for the DOA estimation of broadband signals using the bilinear transformation approach is described. Detailed designs of various modules are in progress. The architecture will be optimized for real time applications.

Chapter 7

COMPUTER SIMULATION

In order to demonstrate the performance of MUSIC algorithm for narrow-band signals, an eight element linear equispaced array of omnidirectional sensors was used with a spacing between the elements equal to 1/2 the propagation wavelength λ. For the case of broad-band signals, this array is used with eight delayed outputs to perform the (BASS-ALE) estimation [29]. The sources were assumed in far field, so source location reduced to azimuth angle θ, measured relative to array broadside. Autoregressive moving average processes (ARMA) have been chosen to represent the contrasting possibilities of spectra with narrow-band and broad-band features. The data record length for all simulations was N (= 1000, 4800) data samples, and the SNR at each sensor is 10 db for all sources.

7.1 Data generation and simulations for narrow band signals

The incoming signals are assumed to be narrow band, that is, their spectrum is zero outside an interval (ω_1 , ω_2) where (ω_2 - ω_1) is small compared with a center frequency ω_o. Thus, an incoming signal can be written as

$$s(t) = i(t) \cos(\omega_o t) - q(t) \sin(\omega_o t) \qquad (7.1)$$

where i(t) and q(t) are low-pass stationary processes, called the in-phase and quadrature-phase components respectively. For such a signal, it is often more convenient to use the complex representation in which s(t) = Re (z(t)), where z(t) is a complex process given by

$$z(t) = s(t) + j\, r(t) \qquad (7.2)$$

In Equation (7. 2), r(t) is a stationary process given by

$$r(t) = q(t) \cos(\omega_o t) + i(t) \sin(\omega_o t) \qquad (7.3)$$

Let w(t) be a complex process given by

$$w(t) = i(t) + j\, q(t) \qquad (7.4)$$

thus

$$z(t) = w(t) \exp(j \omega_o t)$$

$$= s(t) + j\, r(t) \qquad (7.5)$$

and

$$s(t) = \mathrm{Re}\,(z(t)) \qquad (7.6)$$

Now for d incoming signals, the complex signal output of the k^{th} sensor at time t, can be written as

$$y_k(t) = \sum_{i=1}^{d} z_i\,(t - \tau_k(\theta_i)) + n_k(t) \exp(j \omega_o t) \; ; k=1,2, \ldots, m \qquad (7.7)$$

where m is the number of sensors , $\tau_k(\theta_i)$ is the propagation delay between a reference point and the k^{th} sensor for the i^{th} waveform impinging on the array from direction θ_i, and $n_k(t) \exp(j \omega_o t)$ is the added measurement noise.

By noting that the narrow-band assumption implies

$$z_i\,(t - \tau_k(\theta_i)) = z_i\,(t) \exp(-j \omega_o \tau_k(\theta_i)) \qquad (7.8)$$

$y_k(t)$ can be rewritten as

$$y_k(t) = \sum_{i=1}^{d} z_i\,(t) \exp(-j \omega_o \tau_k(\theta_i)) + n_k(t) \exp(j \omega_o t)$$

$$= \sum_{i=1}^{d} w_i(t) \exp(j\omega_o(t-\tau_k(\theta_i))) + n_k(t) \exp(j\omega_o t) \qquad (7.9)$$

Let $x_k(t)$ be the signal obtained by multiplying every $y_k(t)$ by $\exp(-j\omega_o t)$ (demodulation of the incoming signals), such as

$$x_k(t) = \sum_{i=1}^{d} w_i(t) \exp(-j\omega_o \tau_k(\theta_i)) + n_k(t) \qquad (7.10)$$

For simulation purposes, samples of the complex process w(t), and n(t) are generated as follows:

Independent Gaussian noise sequences e(n) are passed through a simple linear system, as shown in Figure.1, with a transfer function given by

$$H_1(z) = \frac{K(1-z^{-2})}{(1-p^2 z^{-2})} \qquad (7.11)$$

where p= .99 j , and K is a constant chosen such that $|H_1(z)| = 1$. The output to the input ratio can be written as

$$\frac{S(z)}{E(z)} = \frac{A(1-z^{-2})}{(1-p^2 z^{-2})} \qquad (7.12)$$

where E(z) and S(z) are the z-transform of the input e(n) and output s(n), respectively. The z-transform S(z) of the unknown sequence s(n) satisfies the algebraic equation

$$S(z) = P^2 z^{-2} S(z) + A(E(z) - z^{-2} E(z)) \qquad (7.13)$$

The response s(n) can thus be determined by taking the inverse z-transform of S(z) to get the recursion or difference equation

$$s(n) = p^2 s(n-2) + A\, e(n) - A\, e(n-2) \qquad (7.14)$$

The time series s(n) at the output of the linear system (narrow-band filter) are multiplied respectively by $\cos(n\pi T/2)$ and $\sin(n\pi T/2)$, where T is the sampling frequency, and then passed through a low pass filter with transfer function

$$H_2(z) = \frac{B(1+z^{-1})}{(1-.99z^{-1})} \qquad (7.15)$$

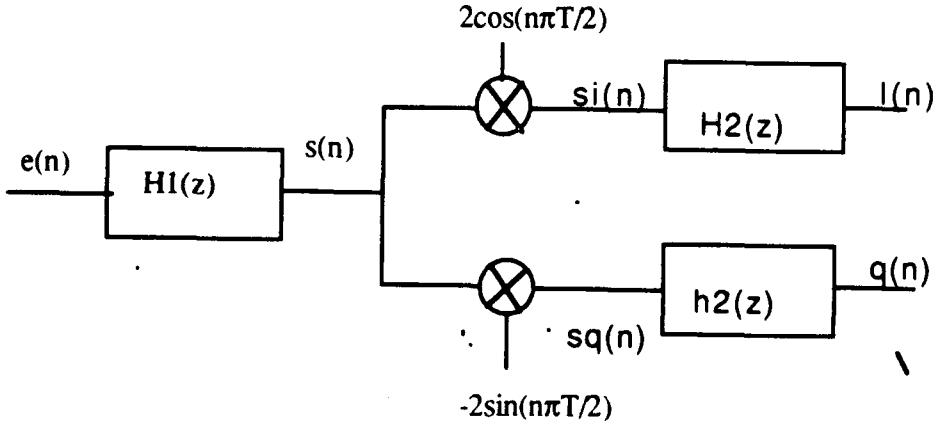


Figure 1. In-phase and quadrature components

The in-phase and quadrature components of the incoming signals are obtained, as shown in Figure 1, as

$$\frac{I(z)}{S^{I}(z)} = \frac{B(1+z^{-1})}{(1-.99z^{-1})} \tag{7.16}$$

and

$$\frac{Q(z)}{S^{q}(z)} = \frac{B(1+z^{-1})}{(1-.99z^{-1})} \tag{7.17}$$

Now by taking the inverse z-transform of the above equations, the recursion equations for I(n) and q(n) are given respectively by

$$I(n) = .99I(n-1) + B\,si(n) + B\,si(n-1) \tag{7.18}$$

$$q(n) = .99I(n-1) + B\,sq(n) + B\,sq(n-1) \tag{7.19}$$

The complex time series s(n) = I(n)+jq(n) are multiplied respectively by $\exp(-j\omega_o \tau_k(\theta_i))$ to simulate the arriving signal from direction θ_i at the k^{th} sensor, and a sample covariance matrix $\mathbf{R}_{xx}$ can be constructed as described in Chapter 1.

For the particular case of a linear array , where the spacing between the elements is $\Delta=\lambda/2$, $\exp(-j\omega_o \tau_k(\theta_i))$ reduces to $\exp(-j\pi(k-1)\sin(\theta_i))$. For simulation purposes, also an infinite sample size covariance matrix was generated by assuming that all sources were mutually uncorrelated and that

$$E(w_i(n)) = 0 \tag{7.20}$$

$$V(w_1(n)) = \sigma_1^2 \quad ; l=1,\ldots,d \tag{7.21}$$

$$E(n_k(n)) = 0 \tag{7.22}$$

$$V(n_k(n)) = \sigma_2^2 \quad ; k=1, \ldots, m \tag{7.23}$$

Under these conditions, it can be shown that the infinite covariance matrix can be obtained as

$$\mathbf{R}_{xx} = \begin{vmatrix} r(0) & r(1) & r(2) & \ldots & r(m-1) \\ r^*(1) & r(0) & r(1) & \ldots & r(m-2) \\ r^*(2) & r^*(1) & r(0) & \ldots & r(m-3) \\ & & & & \\ & & & & \\ r^*(m-1) & r^*(m-2) & r^*(m-3) & \ldots & r(0) \end{vmatrix}$$

where

$$r(k) = \sigma_1^2 \left(\sum_{i=1}^{d} \exp(-j\pi(k-1)\sin(\theta_i)) \right) \; ; \; k = 1,2,\ldots,m \tag{7.24}$$

and

$$r(0) = d\,\sigma_1^2 + \sigma_2^2 \qquad (7.25)$$

Once the data covariance matrix has been derived, the eigendecomposition was performed using Householder's transformations and QR algorithm, and the spatial spectra as function of θ was plotted. It can be noted that for the case of the ideal covariance matrix, the accuracy on the estimation of the arrival angles depends mainly upon the number of iterations performed by the QR algorithm as shown in Figure 2 (a)-(c). By increasing the number of iterations, we have the ability to resolve closely spaced sources. keeping the number of iterations small may yield to inferior results. This is due to the fact that the eigenvalues and eigenvectors may indicate some bias from the theoretical ones. Consequently, the vectors spanning the signal subspace are not orthogonal to the vectors spanning the noise subspace. However, by using Gram-Schmidt orthogonalization [42] to make these subspaces orthogonal, good results may be achieved with small number of iterations as shown in Figure 3 (a)-(c). For the case of finite data covariance matrix, the MUSIC algorithm may indicate some bias in locating the sources even for very large number of iterations, and for several trials. That is, the MUSIC spectrum did not exhibit two peaks at 40° and 45° as shown in Figure 4 (a) and 4 (b) . Moreover, the results indicate that sometimes , the MUSIC algorithm failed completely in locating any source in the interval $[35^{\circ}, 50^{\circ}]$. However, using Gram-Schmidt orthogonalization as for the case of infinite covariance matrix, it was possible to discriminate the targets with small number of iterations as shown in Figure 5 (a) and 5 (b).

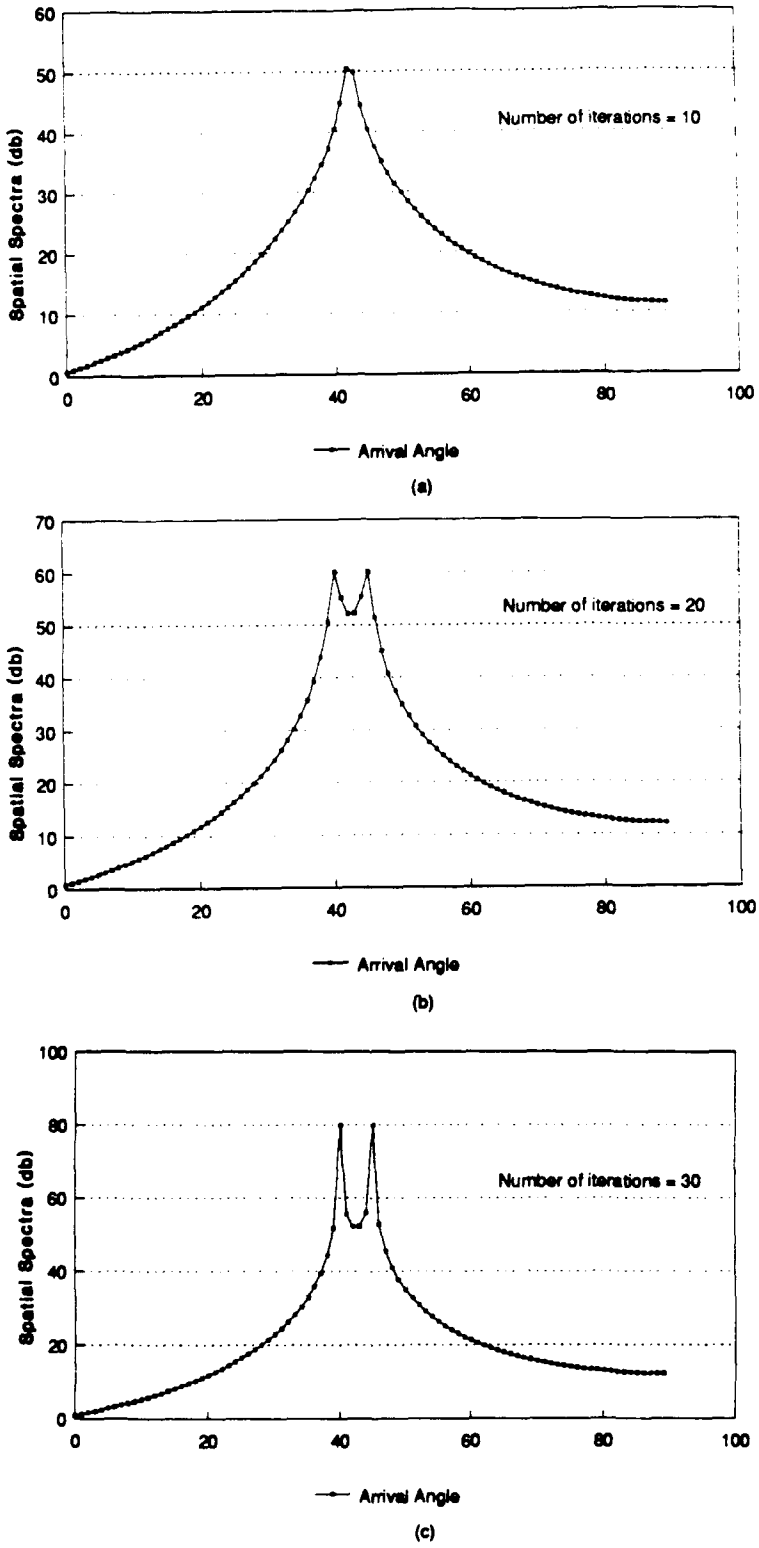


Figure. 2. Spatial spectra without orthogonalization

(Ideal covariance matrix, S/N = 10 db)

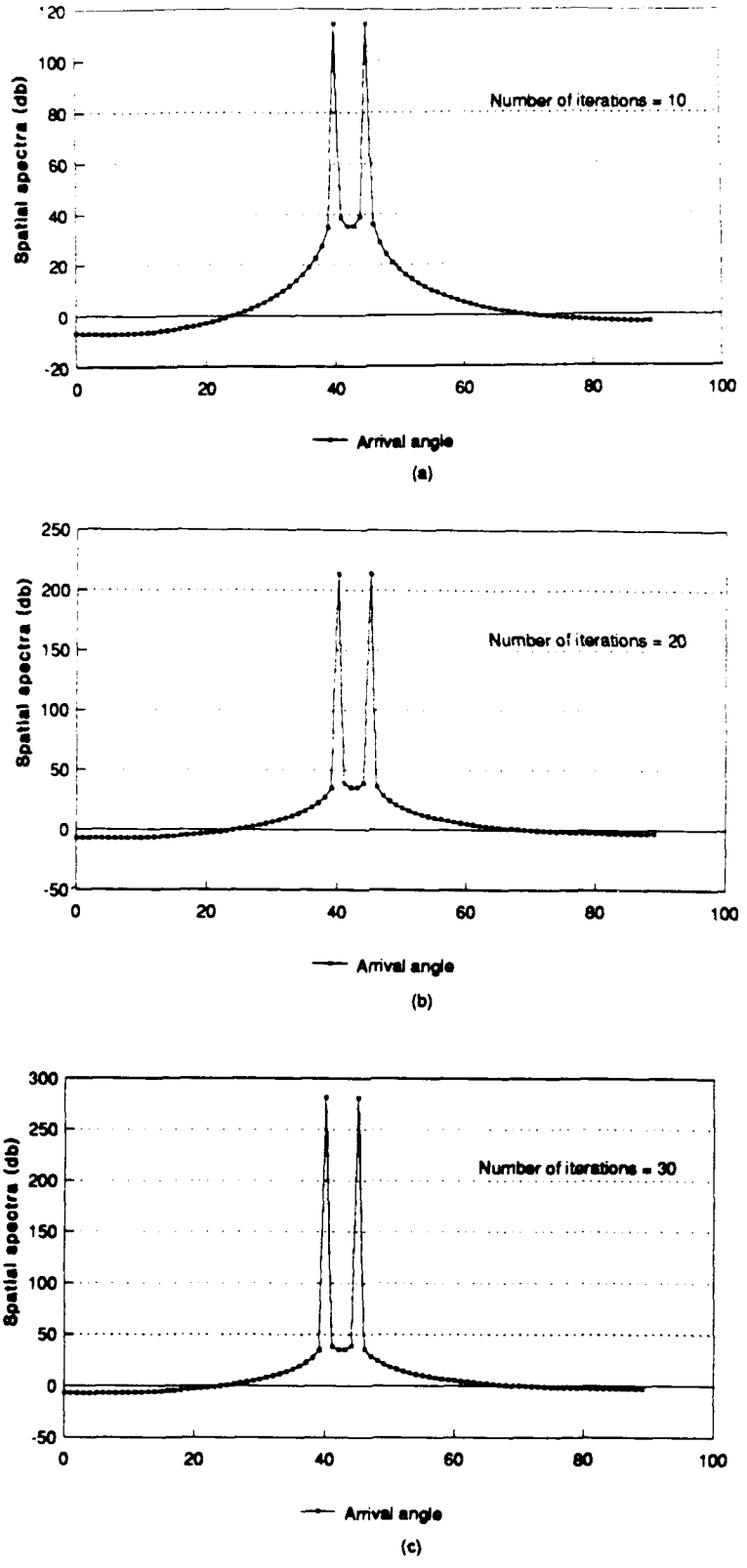
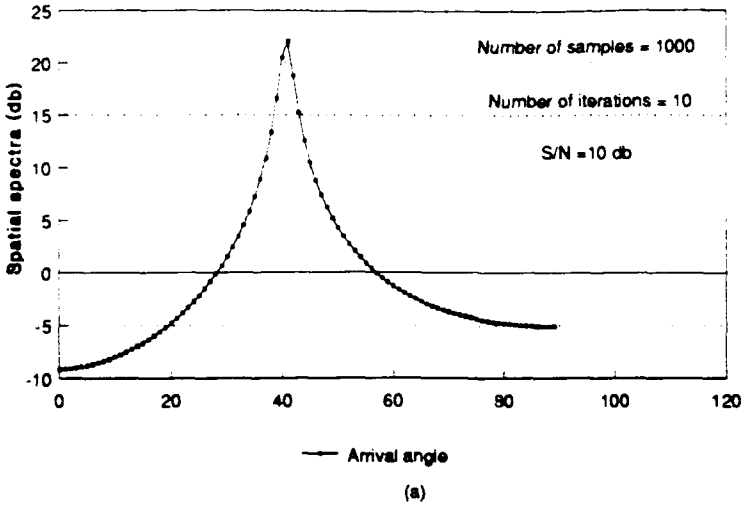
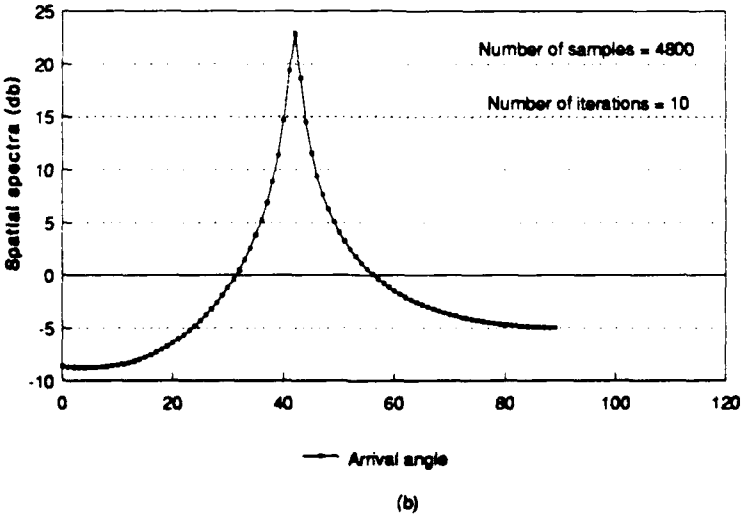


Figure. 3. Spatial spectra with orthogonalization

(Ideal covariance matrix, S/N = 10 db)



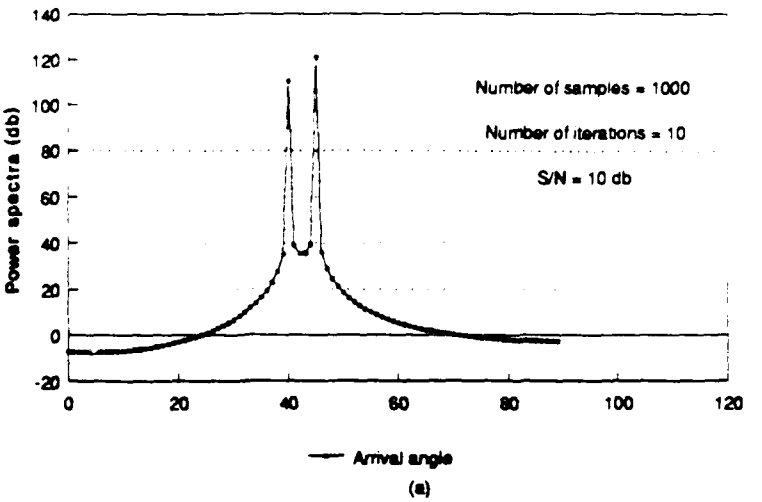
(a)



(b)

Figure. 4. Spatial spectra without orthogonalization

(Finite covariance matrix. S/N = 10 db)



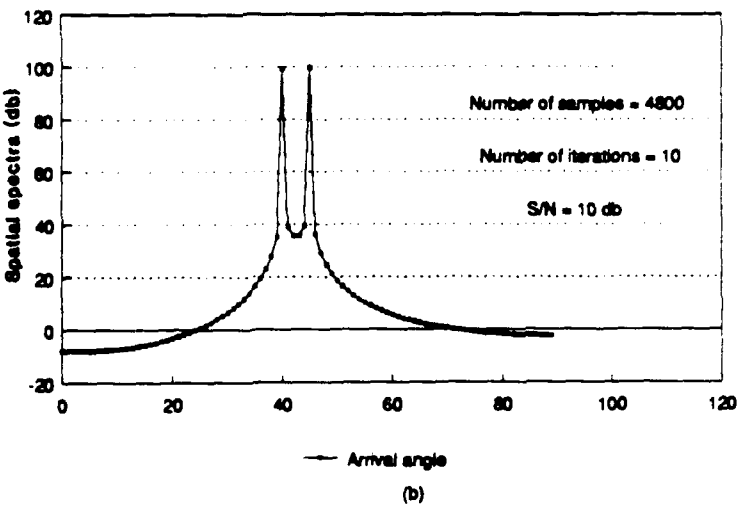


Figure. 5. Spatial spectra with orthogonalization

(Finite covariance matrix , S/N = 10 db)

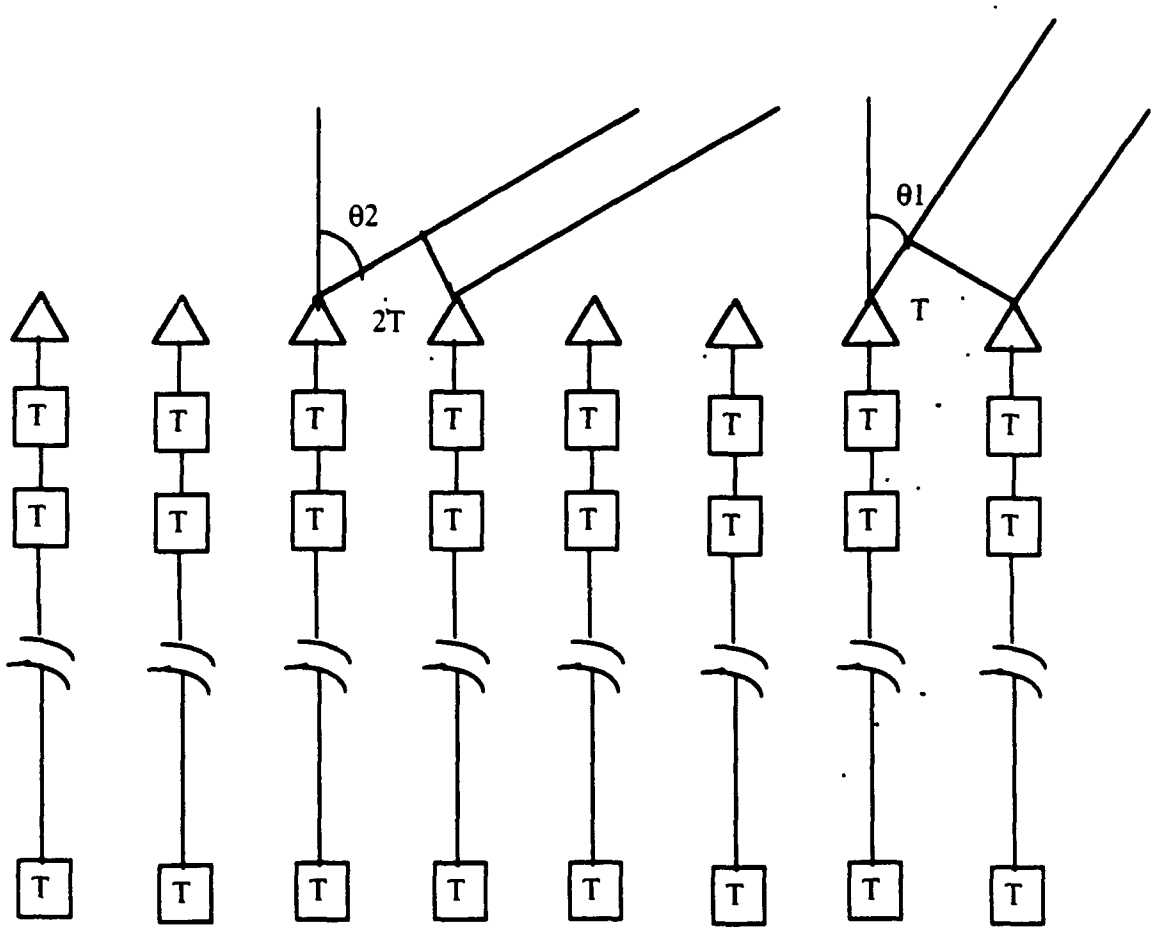
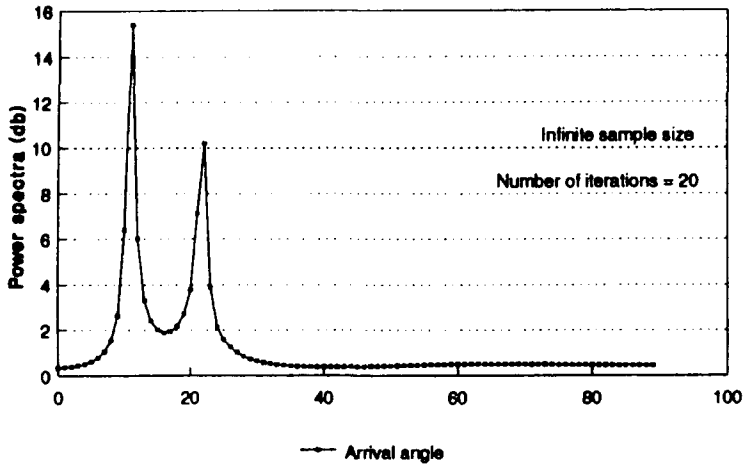
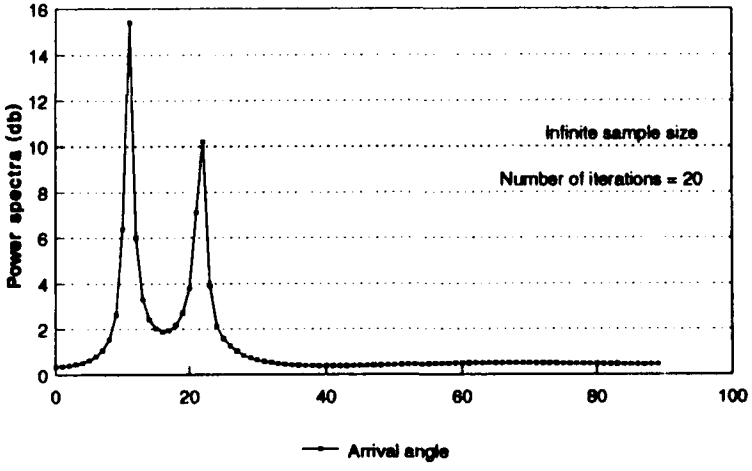
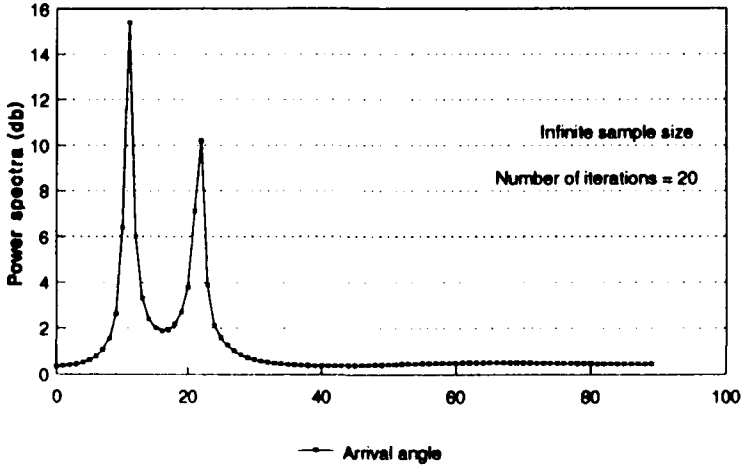


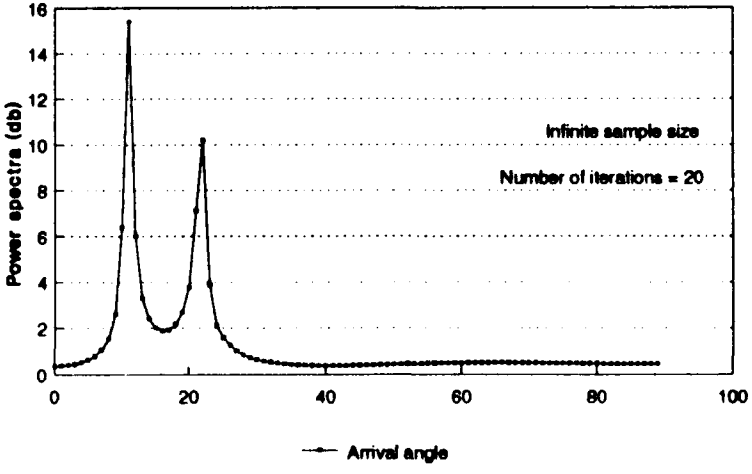
Figure. 6. M array with N delays

(sources arriving at angle θ_1 and θ_2)


Power spectra (db)
Infinite sample size
Number of iterations = 20
Arrival angle


Power spectra (db)
Infinite sample size
Number of iterations = 20
Arrival angle


Power spectra (db)
Infinite sample size
Number of iterations = 20
Arrival angle


Power spectra (db)
Infinite sample size
Number of iterations = 20
Arrival angle

7. 2 . Simulation results for broad-band signals

The emitter signals are generated by passing two independent Gaussian white noise processes through a second order filter having poles z_1 and z_1^* ($z_1 = a \exp(j\theta)$), and two zeros at z= 0 such that

$$H(z) = \frac{K z^2}{(z - z_1)(z - z_1^*)} \tag{7.26}$$

K is determined so that $|H(z)| = 1$, or

$$K = (1-a)\sqrt{1 + a^2 - 2a\cos\theta} \tag{7.27}$$

This band-pass filter is chosen such that its center frequency is equal to $f_o = f_s/10$ and thus θ is given by

$$\theta = (2f_o/f_s)*180 = 36^o \tag{7.28}$$

for a= .85 we have

$$H(z) = \frac{S(z)}{E(z)} = .164 \quad \frac{z^2}{(z^2 - 1.37\, z + .723)} \tag{7.29}$$

The response s(n) can be determined, as in the narrow-band case, by taking the inverse z-transform of S(z) to get the recursion or difference equation

$$s(n)=1.37\ s(n-1)\ -.723\ s(n-2)\ +.164\ e(n) \qquad (7.30)$$

For simulation results, two sources, as shown in Figure 6 , are considered arriving at angle θ_1 and θ_2 such that

$$\tau_k(\theta_1) = (k-1)\ T \qquad (7.31)$$

and

$$\tau_k(\theta_2) = 2(k-1)\ T \qquad (7.32)$$

where T is the sampling period. The values of θ_1 and θ_2 can be determined by letting k=2 in Equations (7.30) and (7.31), or

$$\tau_1(\theta_1) = \Delta \sin\theta_1 / c \qquad (7.33)$$

$$\tau_1(\theta_2) = \Delta \sin\theta_2 / c \qquad (7.34)$$

now, from the fact that $\Delta = \lambda/2$, and the sensor spacing was 1/2 the propagation wavelength for the processing frequency f_o, θ_1 and θ_2 can be shown to be equal to 11.53^{o} and 23.57^{o}. These samples are used to generate the covariance matrix, as described in [29], in order to estimate the spatial spectrum using (BASS-ALE) estimation.

Also an infinite data covariance matrix can be generated by considering two sources having a spectrum and an autocorrelation function, given respectively by

$$S_s = \sigma_1^2 P_{w_o} (\omega - \omega_c) \tag{7.35}$$

and

$$R_s = \sigma_1^2 \frac{\omega_o}{\pi} (\text{sinc}(\omega_o \tau) \exp(j\omega_c \tau) \tag{7.36}$$

For this case , it can be shown that the infinite covariance matrix can be obtained as

$$\mathbf{R}_{xx} = \begin{vmatrix} r(0) & r(1) & r(2) & \dots & r(nm-1) \\ r^*(1) & r(0) & r(1) & \dots & r(nm-2) \\ r^*(2) & r^*(1) & r(0) & \dots & r(nm-3) \\ & & & & \\ r^*(nm-1) & r^*(nm-2) & r^*(nm-3) & \dots & r(0) \end{vmatrix}$$

where m is the number of sensors, and n is the number of delays, and where

$$r(k) = \sigma_1^2 (\text{sinc}(\frac{a\pi}{25}) \exp(j\pi a/5) + \text{sinc}(\frac{b\pi}{25}) \exp(j\pi b/5)) \tag{7.37}$$

with

$a = i+j-2$,

$b = i+2j-3$, and

$k= 2 (i-1) +j \quad ; j=1,m , i=1,n$

and

$$r(0) = 2 \sigma_1^2 + \sigma_2^2 \tag{7.38}$$

An estimate of the DOA's for this case is shown in Figure. 7 .

Chapter 8

CONCLUSIONS

The work performed for the development of parallel architecture for sensor array algorithms for the last one year has been described in Section 8.1 and future work is given in Section 8.2. Further results, recommendations, suggestions and conclusions will be presented in the final report.

8.1: WORK PERFORMED

1. A literature survey has been performed and investigated for various algorithms available for narrow band and wide band cases.

2. In the area of narrow band, MUSIC and ESPRIT algorithms were selected and further studied. These algorithm were converted into computationaly efficient algorithms and subsequently parallelized. Three different architectures namely Systolic architecture, Cordic processors and SIMD are under consideration.

3. These algorithms required eigenvalue decomposition. Householder transformation was used to convert covariance matrix into tridiagonal matrix. The QR method was used to finally obtain eigenvalues and eigenvectors. The detailed parallel architecture has been developed for parallelized Householder transformations and QR method.

4. An architecture for a generalized processing element suitable for sensor array processing elements has been developed. Its custom instruction set has been developed.

5. Single instruction multiple data (SIMD) type of architecture lend themselves for the implementation of narrow band DOA estimation. The

computation of covariance matrix, Householders transformation and QR method con be easily computed using SIMD machine. The work on SIMD machine is under progress.

6. In the case of wideband DOA estimations, various algorithms available in the literature were studied. It was found that wideband DOA estimation is more computationaly intensive than narrow band case. An algorithm proposed by Shaw has been selected for further study. This algorithm again has been modified and substituted with computationally efficient operations. An architecture for the computation of this algorithm is developed.

7. DOA estimation for wide band sources using "Broad-Band Signal-Subspace Spatial Spectral (BASS-ALE) estimation algorithm has been studied, simplified, parallelized and its architecture has also been developed.

8. To verify the validity of all these work, following simulation programs are written.

(a) Data generation of narrowband and wideband sources for eight sensor arrays has been computed.

(b) Simulation of MUSIC algorithm and DOA estimation is completed.

(c) Simulation of BASS-ALE algorithm for wideband sources is almost complete.

(d) Assembly language simulation using DSP56000 DSP processor for MUSIC algorithm is in progress.

(e) Simulation of DOA estimation using bilinear transformation approach is almost complete.

8.2 FUTURE WORK

1. The pipelined architecture of MUSIC should be developed using Application Specific Integrated Circuit (ASIC) chips. This will include processing modules, latches, buffers and memory elements.

2. A simulation can be performed at architecture level using VHDL software to check computational complexity of the entire parallel architecture. Logical level and circuit level design can be done using Viewlogic's computer aided design (CAD) tool Workview 4.0. This circuit level design can be converted into Very Large Scale Integration (VLSI) level design by various design tools available.

3. As the entire architecture cannot be accommodated on a single chip, multichip modules can be used. This will involve study of inter-chip communication, I/O bus architecture for each chip, bus arbitrator etc.

4. A single Generalized Processor (GP) has been developed which minimizes cost and design time can be converted into a RISC processor. This RISC processor will simplify the design of control unit, enhance the speed of the operation by improved pipelining.

5. The entire MUSIC algorithm can be executed on several RISC processors in a miltiprocessor environment depending on the speed requirement.

6. This generalized RISC processor can not only be used for MUSIC algorithm, but also can be used for ESPRIT and other sensor array processing algorithms and digital signal processing algorithms.

7. The MUSIC/ESPRIT algorithm will be modified in an attempt to obtain more accurate DOA's.

8. A detailed architecture for ESPRIT algorithm is being developed and can also be extended to wideband case.

9. A study will be performed to estimate real time requirements for the computations of DOA. Based on this study, real time architecture for the computation of MUSIC and ESPRIT will be proposed.

10. Parallel architecture for BASS-ALE algorithm for wide band sources has been described and requires completion of the following:

(a) Complete the design of signal-subspace dimension D estimate unit.

(b) The design of power method (wideband case) unit.

(c) VLSI design or design using commercially available DSPs or combination of the two approaches need to be done.

(d) Low level simulation of this method need to be performed as high level simulation using FORTRAN is already completed.

11. Parallel architecture for bilinear transformation algorithm for wide band sources, has also been designed and needs the design of the following:

(a) The design will be optimized for maximum parallelism at every level. This will enable the architecture to be sufficient for higher sampling rates.

(b) The various modules will be designed in detail, with emphasis on timing requirements at each stage and the routing of data.

(c) The modules will be implemented using commercially available DSP simulation software or ASIC. The architecture will again be simulated functionally using either DSP software or VHDL software.

(d) As customized processing elements for this class of algorithm has been designed, the next step will be its VLSI implementation and development of its assembler or other software.

12. The two algorithms for the DOA estimation for wideband sources namely BASS-ALE and bilinear transformation will be compared based on hardware complexity, computation time requirements an accuracy of results. One of the algorithm for the wideband case will be selected and will be designed completely.

APPENDIX

```
c*************************************************************************
c       This program implements the MUSIC algorithm for the direction  *
c       of arrival (DOA's) estimation for narrow band signals. Both     *'
c       incoherent and partially coherent signals are considered.       *
c       The data covariance matrix is generated by considering the      *
c       time series at the output of a narrow-band filter whose input   *
c       is a white noise of power sigma1.                               *
c       The array chosen has a maximum of eight elements and the        *
c       spacing between two elements has been chosen to be equal to     *
c       Lamda/2.                                                        *
c                                                                       *
c       Also an infinite sample size data covariance was genrated       *
c       to estimate the DOA's of incoherent sources.                   *'
c                                                                       *
c*************************************************************************

c       c =(cr,ci)    _Complex data covariance matrix
c       nc            _Number of sensors
c       ns            _Number of sources
c       z =(zr,zi)    _Complex array of dimension ns representing
                       the sources
c       th            _Array of dimension ns representing the
                       angles of arrival of the sources
c       x =(xr,xi)    _Complex array of dimension nc representing
                       the sensors outputs
c       n =(nr,ni)    _Complex array of dimension nc representing
                       the added measurement noise at the sensors
c       r =(rr,ri)    _Complex matrix representing the data covariance
                       matrix after Householder's transformations'
c       t =(tr,ti)    _Diagonal matrix resulting from the QR
                       transformations
c       u =(ur,ui)    _Complex matrix whose rows are the eigenvectors
                       of the original data covariance matrix
c       pm            _Array representing the spatial spectrum
c       sig1          _Signal power
c       Sig2          _Noise  power


        character out1*10,out2*10,out3*10,out4*10
        real cr(8,8), ci(8,8), zr(8), zi(8)
        real xr(8),xi(8),nr(8), ni(8),th(8)
        real yr(8,8),yi(8,8),tr(8,8),ti(8,8)
        real rr(8,8),ri(8,8),ur(8,8),ui(8,8),pm(361)
        integer option
        pi= acos(-1.)
        print *,'Input number of sensors'
        read *, nc
        print *,'Input number of signals less than number of sensors'
        read  *, ns
        print*, 'Input the angle of arrival for each signal (in degrees)'
        do 1 k=1,ns
        print*, 'th(',k,')=?'
        read  *, th(k)
        th(k)= pi*sin(th(k)* pi/180.)/2.
 1      continue
        print *,' Input the signal power'
        read *, sig1
        print *,' Input the noise  power'
        read *, sig2
```

```
      print *,'Input 0 for infinite sample size, 1 otherwise'
      read * ,option
      if (option.eq.0) goto 40
      print *, 'Input number of samples'
      read  *,n
      print *,'0 for partially coherent, 1 otherwise'
      read *,option
      if (option.eq.0) then
      out1="Co-cov"
      out2="Co-hh"
      out3="Co-qr"
      out4="Co-pow"
      else
      out1="Ic-cov"
      out2="Ic-hh"
      out3="Ic-qr"
      out4="Ic-pow"
      endif
      nn=n/2
      sig1=10.
      sig2=1.
      seed=rnd(0.0)
      stdev1=sqrt(sig1)
      stdev2=sqrt(sig2)
      do 2 j=1,nc
      do 2 l=j,nc
      cr(j,l)=0.
      ci(j,l)=0.
2     continue
      do j=1,nc
      do k=2,3
      sx(j,k)=0.
      sy(j,k)=0.
      nx(j,k)=0.
      ny(j,k)=0.
      enddo
      sc(j,2)=0.
      ss(j,2)=0.
      si(j,2)=0.
      sq(j,2)=0.
      nc(j,2)=0.
      ns(j,2)=0.
      ni(j,2)=0.
      nq(j,2)=0.
      enddo
      do 3 i=1,n
      call gauss(stdev1,x,seed)
      do j=1,ns
      if(option.eq.0.and.i.le.nn) goto 30
      call gauss(stdev1,x,seed)
30    sy(j,1)=-.9839*sy(j,3)+(1./100.5)*(x-sx(j,3))
      sc(j,1)=sy(j,1)*cos(float(i)*.5 *pi)
      ss(j,1)=sy(j,1)*sin(float(i)*.5 *pi)
      si(j,1)=.99*si(j,2)+.5*(sc(j,2)+sc(j,1))
      sq(j,1)=.99*sq(j,2)+.5*(ss(j,2)+ss(j,1))
      zr(j)=si(j,1)
      zi(j)=sq(j,1)
      sy(j,3)=sy(j,2)
      sx(j,3)=sx(j,2)
```

```
      sy(j,2)=sy(j,1)
      sx(j,2)=x
      ss(j,2)=ss(j,1)
      sc(j,2)=sc(j,1)
      si(j,2)=si(j,1)
      sq(j,2)=sq(j,1)
      enddo

      do j=1,nc
      call gauss(stdev2,x,seed)
      ny(j,1)=-.9839*ny(j,3)+(1./100.5)*(x-nx(j,3))
      nc(j,1)=ny(j,1)*cos(float(i)*.5 *pi)
      ns(j,1)=ny(j,1)*sin(float(i)*.5 *pi)
      ni(j,1)=.99*ni(j,2)+.5*(nc(j,2)+nc(j,1))
      nq(j,1)=.99*nq(j,2)+.5*(ns(j,2)+ns(j,1))
      nor(j)=ni(j,1)
      noi(j)=nq(j,1)
      ny(j,3)=ny(j,2)
      nx(j,3)=nx(j,3)
      ny(j,2)=ny(j,1)
      nx(j,2)=x
      ns(j,2)=ns(j,1)
      nc(j,2)=nc(j,1)
      ni(j,2)=ni(j,1)
      nq(j,2)=nq(j,1)
      enddo

      do 4 j=1,nc
      xr(j)=0.
      xi(j)=0.
      do 7 k=1,ns
      xr(j)=xr(j)+zr(k)*cos(float(j-1)*th(k))
      xr(j)=xr(j)+zi(k)*sin(float(j-1)*th(k))
      xi(j)=xi(j)-zr(k)*sin(float(j-1)*th(k))
      xi(j)=xi(j)+zi(k)*cos(float(j-1)*th(k))
7     continue
      xr(j)=xr(j)+nor(j)
      xi(j)=xi(j)+noi(j)
4     continue

c
c     Generate the data covariance matrix for finite
c     sample size
c
      do 9 j=1,nc
      do 8 l=j,nc
      cr(j,l)=cr(j,l)+xr(j)*xr(l)+xi(j)*xi(l)
      ci(j,l)=ci(j,l)+xi(j)*xr(l)-xi(l)*xr(j)
8     continue
3     continue
      do 9 j=1,nc
      do 9 l=j,nc
      cr(j,l)=cr(j,l)/float(n)
      cr(l,j)=cr(j,l)
      ci(j,l)=ci(j,l)/float(n)
      ci(l,j)=-ci(j,l)
9     continue
      goto 50
c
```

```
c       Generate the data covariance matrix for infinite
c       sample size
c
40      do 10 j=1,nc
        do 10 l=j,nc
        cr(j,l)=0.
        a=float(j-l)
        do 11 k=1,ns
        c1=cos(a*th(k))
        s1=sin(a*th(k))
        cr(j,l)= cr(j,j)+c1*sig1
        ci(j,l)= ci(j,l)+s1*sig1
11      continue
        cr(l,j)=cr(j,l)
        ci(l,j)=-ci(j,l)
10      continue
        do 12 i=1,nc
        cr(i,i)=cr(i,i)+sig2
12      continue
50      print    *, 'Data covariance matrix'
        do i=1,nc
        print    *  , (cr(i,j),j=1,nc)
        enddo
        print *,    '                '
        do i=1,nc
        print    *  , (ci(i,j),j=1,nc)
        enddo
        call hhc (cr,ci,rr,ri,ur,ui,nc)
        print    *, 'HH matrix'
        do i=1,nc
        print    *  , (rr(i,j),j=1,nc)
        enddo
        print *,    '                '
        do i=1,nc
        print    *  , (ri(i,j),j=1,nc)
        enddo

        call qrc (rr,ri,tr,ti,ur,ui,nc)
        print    *, 'QR matrix'
        do i=1,nc
        print    *  , (tr(i,j),j=1,nc)
        enddo
        print *,    '                '
        do i=1,nc
        print    *  , (ti(i,j),j=1,nc)
        enddo

        call power (ur,ui,pm,ns,nc)
        do i=1,90
        jj=i-1
        print *,    jj,pm(i)
        enddo
        stop
        end


c
c       subroutine name : Gauss
c
```

```
c       This subroutine generate two gaussian random numbers
c
c       Input parameters
c          stdev  _standart deviation of white gaussian noise
c          seed   _arbitrary integer which will serve as seed
c                  for generating a gausian number
c
c       Output parameters
c          a,b    _two gaussian numbers denoting the in-phase
c                  and quadrature components of the white noise
c
        subroutine gauss(sdev,a,seed)
        real sdev,a
        s=0.
        do 1 ii=1,12
        seed=rnd(seed)
        s=s+seed
1       continue
        a=(s-6.)*sdev
        return
        end

c
c     subroutine name: hhc
c
c     This subroutine reduces a full dense hermetian matrix
c     to tridiagonal one using Housholder,s tranformations
c
c     Input parameters
c         c=(cr,ci)     _data covariance matrix
c         n             _matrix order
c
c     Output parameters
c         r=(rr,ri)     _tridiagonal matix
c         u=(ur,ui)     _product of the (n-2) Householder's'
c                        transformations, also partial result
c                        of the eigenvectors.
c
      subroutine hhc (cr,ci,rr,ri,ur,ui,n)
      real rr(8,8),ri(8,8),ur(8,8),ui(8,8),wr(7),wi(7)
      real cr(8,8),ci(8,8)
c
c     Initialisation for the eigenvectors
c
      do 1 i=1,n
      do 2 j=1,n
      ur(i,j)=0.0
      ui(i,j)=0.0
2     continue
1     continue
      do 3 i=1,n
      ur(i,i)=1
      ui(i,i)=0
3     continue
c
c     Compute the Householder's transformations'
c
      do 4 i=1,n-2
      r1=0.0
```

```
      do 5 j=i+1,n
      r1=r1+cr(j,i)*cr(j,i)+ci(j,i)*ci(j,i)
5     continue
      d=sqrt(cr(i+1,i)*cr(i+1,i)+ci(i+1,i)*ci(i+1,i))
      r1=sqrt(r1)/d
      wr(i)=cr(i+1,i)+r1*cr(i+1,i)
      wi(i)=ci(i+1,i)+r1*ci(i+1,i)
      cr(i+1,i)=-r1*cr(i+1,i)
      ci(i+1,i)=-r1*ci(i+1,i)
      cr(i,i+1)=cr(i+1,i)
      ci(i,i+1)=-ci(i+1,i)
      do 6 j=i+1,n-1
      wr(j)=cr(j+1,i)
      wi(j)=ci(j+1,i)
6     continue
      c=0.
      do 18 j=i,n-1
      c=c+wr(j)*wr(j)+wi(j)*wi(j)
18    continue
      c=c/2
c
c     Compute the update covariance data matrix for every
c     transformation
c
      do 7 j=i+2,n
      cr(i,j)=0.0
      cr(j,i)=0.0
      ci(i,j)=0.0
      ci(j,i)=0.0
7     continue
      do 8 j=i+1,n
      d1=0.0
      d2=0.0
      do 9 k=i+1,n
      d1=d1+wr(k-1)*cr(k,j)+wi(k-1)*ci(k,j)
      d2=d2+wr(k-1)*ci(k,j)-wi(k-1)*cr(k,j)
9     continue
      d1=d1/c
      d2=d2/c
      do 10 k=i+1,n
      cr(k,j)=cr(k,j)-(wr(k-1)*d1-wi(k-1)*d2)
      ci(k,j)=ci(k,j)-(wr(k-1)*d2+wi(k-1)*d1)
10    continue
8     continue
      do 11 j=i+1,n
      d1=0.0
      d2=0.0
      do 12 k=i+1,n
      d1=d1+wr(k-1)*cr(j,k)-wi(k-1)*ci(j,k)
      d2=d2+wr(k-1)*ci(j,k)+wi(k-1)*cr(j,k)
12    continue
      d1=d1/c
      d2=d2/c
      do 13 k=i+1,n
      cr(j,k)=cr(j,k)-(d1*wr(k-1)+d2*wi(k-1))
      ci(j,k)=ci(j,k)-(d2*wr(k-1)-d1*wi(k-1))
13    continue
11    continue
c
```

```
c     Compute the product of the Housholder's transformations'
c     which will serve as a partial result for obtaining the
c     eigenvectors of the original matrix
c
      do 14 j=i,n
      d1=0.0
      d2=0.0
      do 15 k=i+1,n
      d1=d1+wr(k-1)*ur(k,j)+wi(k-1)*ui(k,j)
      d2=d2+wr(k-1)*ui(k,j)-wi(k-1)*ur(k,j)
15    continue
      d1=d1/c
      d2=d2/c
      do 16 k=i+1,n
      ur(k,j)=ur(k,j)-(wr(k-1)*d1-wi(k-1)*d2)
      ui(k,j)=ui(k,j)-(wr(k-1)*d2+wi(k-1)*d1)
16    continue
14    continue
4     continue
      do 17 i=1,n
      do 17 j=1,n
      rr(i,j)=cr(i,j)
      ri(i,j)=ci(i,j)
17    continue
      return
      end



c
c     subroutine name: qrc
c
c     This subroutine reduces a tridiagonal matrix to a
c     diagoanl one using the QR algorithm
c
c     Input parameters
c         y=(yr,yi)     _tridiagonal matrix
c         u=(ur,ui)     _product of the (n-2) Householder's'
c                        transformations, also partial result
c                        of the eigenvectors.
c         n             _matrix order
c
c
c     Output parameters
c         t=(tr,ti)     _diagonal matrix whose entries are the
c                        eigenvalues of the original matrix
c
c         u=(ur,ui)     _matrix whose rows are the eigenvalues
c                        of the original matrix
c


        subroutine qrc(rr,ri,tr,ti,ur,ui,n)
        real tr(8,8),ti(8,8),qr(8,8), qi(8,8)
        real rr(8,8),ri(8,8),ur(8,8),ui(8,8)
        real  fr(8,8), fi(8,8)
        do 1 i=1,n
        do 1 j=1,n
        tr(i,j)=rr(i,j)
```

```
        ti(i,j)=ri(i,j)
1       continue
c
c       Initialise the number of iterations needed to perform
c       the eigendecomposition of the tridiagonal matrix
c
        iter=0
15      iter=iter+1
        do 2 i=1,n
        do 2 j=1,n
        qr(i,j)=0
        qi(i,j)=0
2       continue
        do 3 i=1,n
        qr(i,i)=1
        qi(i,i)=0.
3       continue
c
c       Compute the plane rotations of every iteration
c
        y=tr(1,1)
        do 4 i=1,n-1
        x=tr(i+1,i)*tr(i+1,i)+ti(i+1,i)*ti(i+1,i)
        if (x.eq.0.)
        y=tr(i+1,i+1)
        else
        x=x+y*y
        x=sqrt(x)
        pr11=y/x
        pi11=0.
        pr12=tr(i+1,i)/x
        pi12=-ti(i+1,i)/x
        pr21=-pr12
        pi21=pi12
        pr22=pr11
        pi22=0.
c
c       Compute the eigenvectors
c
        do 7 j=1,n
        cr1=pr11*ur(i,j)-pi11*ui(i,j)+pr12*ur(i+1,j)-pi12*ui(i+1,j)
        ci1=pr11*ui(i,j)+pi11*ur(i,j)+pr12*ui(i+1,j)+pi12*ur(i+1,j)
        cr2=pr21*ur(i,j)-pi21*ui(i,j)+pr22*ur(i+1,j)-pi22*ui(i+1,j)
        ci2=pr21*ui(i,j)+pi21*ur(i,j)+pr22*ui(i+1,j)+pi22*ur(i+1,j)
        ur(i,j)=cr1
        ui(i,j)=ci1
        ur(i+1,j)=cr2
        ui(i+1,j)=ci2
7       continue
c
c       Compute the orthogonal matrix Q for every transformation
c
        do 8 j=1,n
        cr1=pr11*qr(i,j)-pi11*qi(i,j)+pr12*qr(i+1,j)-pi12*qi(i+1,j)
        ci1=pr11*qi(i,j)+pi11*qr(i,j)+pr12*qi(i+1,j)+pi12*qr(i+1,j)
        cr2=pr21*qr(i,j)-pi21*qi(i,j)+pr22*qr(i+1,j)-pi22*qi(i+1,j)
        ci2=pr21*qi(i,j)+pi21*qr(i,j)+pr22*qi(i+1,j)+pi22*qr(i+1,j)
        qr(i,j)=cr1
        qi(i,j)=ci1
```

```
        qr(i+1,j)=cr2
        qi(i+1,j)=ci2
8       continue
        j=i+1
        y=pr21*tr(i,j)-pi21*ti(i,j)+pr22*tr(i+1,j)-pi22*ti(i+1,j)
        endif
4       continue
c
c       Compute the update trdiagonal matrix
c
        do 9 i=1,n
        do 9 j=1,n
        rr(i,j)=0
        ri(i,j)=0
9       continue
        do 10 i=1,n
        do 10 j=1,n
        do 10 k=1,n
        rr(i,j)=rr(i,j)+qr(i,k)*tr(k,j)-qi(i,k)*ti(k,j)
        ri(i,j)=ri(i,j)+qi(i,k)*tr(k,j)+qr(i,k)*ti(k,j)
10      continue
        do 11 i=1,n
        do 11 j=1,n
        tr(i,j)=0
        ti(i,j)=0
11      continue
        do 12 i=1,n
        do 12 j=1,n
        do 12 k=1,n
        tr(i,j)=tr(i,j)+rr(i,k)*qr(j,k)+ri(i,k)*qi(j,k)
        ti(i,j)=ti(i,j)+ri(i,k)*qr(j,k)-rr(i,k)*qi(j,k)
12      continue
c
c       Test for convergence.
c       Perform the iterations until sum of the square of the
c       subdiagonal elements are less than some specified
c       value (tolerance)
c

        s=0.
        do 13 i=1,n-1
        j=i+1
        s=s+tr(j,i)*tr(j,i)+ti(j,i)*ti(j,i)
13      continue
        print *,iter
        if (s.gt.0.0001) goto 15
        return
        end


c
c     subroutine name: power
c
c     This subroutine calculates the spatial spectrum using
c     MUSIC algorithm for narrow-band signals
c
c     Input parametres
c         u=(ur,ui)     _matrix whose rows are the eigenvectors
c                        of the original data covariance matrix
```

```
c         n              _matrix order
c         d              _dimension of the signal subspace
c
c     Output parameter
c        pm             _Array representing the spatial spectrum
c


        subroutine power(ur,ui,pm,d,n)
        integer d
        real ur(8,8),ui(8,8),pm(361)
        pi=acos(-1.)
        do 1 i=1,90
        theta=((float(i)-1.)/180.)* pi
        ph= pi*sin(theta)/2.
        pm(i)=0.
        do 2 j=d+1,n
        sr=0.
        si=0.
        do 3 k=1,n
        sr=sr+ur(j,k)*cos((float(k)-1.)*ph)
        sr=sr+ui(j,k)*sin((float(k)-1.)*ph)
        si=si+ur(j,k)*sin((float(k)-1.)*ph)
        si=si-ui(j,k)*cos((float(k)-1.)*ph)
3       continue
        pm(i)=pm(i)+sr*sr+si*si
2       continue
        jj=i-1
        pm(i)=1./pm(i)
        print *,jj, pm(i)
1       continue
        return
        end
```

```
c*************************************************************************
c      This program implements the MUSIC algorithm for the direction  *
c      of arrival (DOA's) estimation for broad-band signals. Both     *'
c      incoherent and partially coherent signals are considered.      *
c      The data covariance matrix is generated by considering         *
c      two broad-band emitters supposed to be present at 11.53        *
C      and 23.57 with identical spectra.The emitter signals are the   *
c      time series at the output of a band pass filter (BPF) whose    *
c      input is a white noise of power sigma1.                        *
c      The BPF has a center frequency f0=2khz. The class of input     *
c      signals expected is bandlimited to below 10khz. The sampling   *
c      frequency is selected as 20khz.                                *
c      The array chosen has a maximum of eight elements and the       *
c      spacing between two elements has been chosen to be equal to    *
c      c/f0, where c is the propagation velocity.                     *
c                                                                     *
c*************************************************************************

c       c =(cr,ci)    _Complex data covariance matrix
c       nc            _Number of sensors
c       ns            _Number of sources (chosen to be equal to two)
c       z =(zr,zi)    _Complex array representing the signals at
                       the sensors and different delays.
c       x1 =(x1r,x1i)_Complex array representing the samples of
                       source number 1.
c       x2 =(x2r,x2i)_Complex array representing the samples of
                       source number 2.
c       n =(nr,ni)    _Complex matrix representing the added
                       measurement noise at the sensors
c       r =(rr,ri)    _Complex matrix representing the data covariance
                       matrix after Householder's transformations'
c       t =(tr,ti)    _Diagonal matrix resulting from the QR
                       transformations
c       u =(ur,ui)    _Complex matrix whose rows are the eigenvectors
                       of the original data covariance matrix
c       pm            _Array representing the spatial spectrum
c       sig1          _Signal power
c       Sig2          _Noise  power

        character out1*10,out2*10,out3*10,out4*10
        real cr(64,64), ci(64,64), zr(64),zi(64)
        real x1r(15),x1i(15),x2r(15),x2i(15),wr(64),wi(64)
        real yr(64,64), yi(64,64), tr(64,64),ti(64,64)
        real rr(64,64),ri(64,64),ur(64,64),ui(64,64),pm(360)
        real nr(3,8),ni(3,8)
        integer option
        tpi=2*acos(-1.)
        print *, 'input the number of samples multiple of 8'
        read  *,n
        n=n/8
        print *,'0 for partially coherent, 1 otherwise'
        read *,option
        if (option.eq.0) then
        out1="CCVdat"
        out2="CHHdat"
        out3="CQRdat"
        out4="CPOdat"
        else
        out1="ICVdat"
```

```
      out2="IHHdat"
      out3="IQRdat"
      out4="IPOdat"
      endif
      sig1=10.
      sig2=1.
      seed=rnd(0.0)
      stdev1=sqrt(sig1)
      stdev2=sqrt(sig2)
      do 1 j=1,64
      do 1 l=j,64
      cr(j,l)=0.
      ci(j,l)=0.
1     continue
c
c     Generate sources arriving from angles 11.53 and 23.57
c     respectively
c
      call gauss(stdev1,x1,x2,seed)
      x1r(1)=.164*x1
      x1i(1)=.164*x2
      if (option.eq.0) goto 12
      call gauss(stdev1,x1,x2,seed)
12    x2r(1)=.164*x1
      x2i(1)=.164*x2
      call gauss(stdev1,x1,x2,seed)
      x1r(2)=.164*x1+1.37*x1r(1)
      x1i(2)=.164*x2+1.37*x1i(1)
      if (option.eq.0) goto 13
      call gauss(stdev1,x1,x2,seed)
13    x2r(2)=.164*x1+1.37*x2r(1)
      x2i(2)=.164*x2+1.37*x2i(1)
      do 2 i=3,15
      call gauss(stdev1,x1,x2,seed)
      x1r(i)=.164*x1+1.37*x1r(i-1)-.723*x1r(i-2)
      x1i(i)=.164*x2+1.37*x1i(i-1)-.723*x1i(i-2)
      if (option.eq.0) goto 14
      call gauss(stdev1,x1,x2,seed)
14    x2r(i)=.164*x1+1.37*x2r(i-1)-.723*x2r(i-2)
      x2i(i)=.164*x2+1.37*x2i(i-1)-.723*x2i(i-2)
2     continue
c
c     Generate added noise of same bandwidth as sources
c
      do j=1,8
      call gauss(stdev2,x1,x2,seed)
      nr(1,j)=.164*x1
      ni(1,j)=.164*x1
      enddo

      do j=1,8
      call gauss(stdev2,x1,x2,seed)
      nr(2,j)=.164*x1 + 1.37*nr(1,j)
      ni(2,j)=.164*x1 + 1.37*nr(1,j)
      enddo

      do j=1,8
      call gauss(stdev2,x1,x2,seed)
      nr(3,j)=.164*x1 + 1.37*nr(2,j) - .723*nr(1,j)
```

```
      ni(3,j)=.164*x1 + 1.37*nr(2,j) - .723*ni(1,j)
      enddo

      do 3 j=1,8
      j1=16-j
      j2=17-2*j
      zr(j)=x1r(j1)+x2r(j2)+nr(3,j)
      zi(j)=x1i(j1)+x2i(j2)+ni(3,j)
3     continue
      do 4 i=1,n
      do 5 k=7,1,-1
      do 6 j=1,8
      j1=j+k*8
      zr(j1)=zr(j)
      zi(j1)=zi(j)
6     continue
      do 7 j=1,14
      x1r(j)=x1r(j+1)
      x1i(j)=x1i(j+1)
      x2r(j)=x2r(j+1)
      x2i(j)=x2i(j+1)
7     continue
      call gauss(stdev1,x1,x2,seed)
      x1r(15)=.164*x1+1.37*x1r(14 )-.723*x1r(13 )
      x1i(15)=.164*x2+1.37*x1i(14 )-.723*x1i(13 )
      if (option.eq.0) goto 15
      call gauss(stdev1,x1,x2,seed)
15    x2r(15)=.164*x1+1.37*x2r(14 )-.723*x2r(13 )
      x2i(15)=.164*x2+1.37*x2i(14 )-.723*x2i(13 )

      do j=1,8
      nr(1,j)=nr(2,j)
      ni(1,j)=ni(2,j)
      nr(2,j)=nr(3,j)
      ni(2,j)=ni(3,j)
      call gauss(stdev2,x1,x2,seed)
      nr(3,j)=.164*x1 + 1.37*nr(2,j) - .723*nr(1,j)
      ni(3,j)=.164*x1 + 1.37*nr(2,j) - .723*ni(1,j)
      enddo

      do 8 j=1,8
      j1=16-j
      j2=17-2*j
      zr(j)=x1r(j1)+x2r(j2)+nr(3,j)
      zi(j)=x1i(j1)+x2i(j2)+ni(3,j)
8     continue
5     continue
c
c     Generate the data covariance matrix (64,64)
c
      do 9 j=1,64
      do 9 l=j,64
      cr(j,l)=cr(j,l)+zr(j)*zr(l)+zi(j)*zi(l)
      ci(j,l)=ci(j,l)+zi(j)*zr(l)-zi(l)*zr(j)
9     continue
4     continue
      do 10 j=1,64
      do 10 l=j,64
      cr(j,l)=cr(j,l)/float(n)
```

```
      cr(1,j)=cr(j,1)
      ci(j,1)=ci(j,1)/float(n)
      ci(1,j)=-ci(j,1)
10    continue
      open(unit=20,file=out1,access='sequential')
      write(20,*) 'Data covariance matrix'
      do i=1,64
      write(20,100) (cr(i,j),j=1,64)
      enddo
      write(20,*) '                '
      do i=1,64
      write(20,100) (ci(i,j),j=1,64)
      enddo
      close (unit=20)
      print *,'1'
      n=64
      call hhc (cr,ci,rr,ri,ur,ui,n)
      open(unit=21,file=out2,access='sequential')
      write(21,*)'Householder matrix'
      do i=1,64
      write(21,100) (rr(i,j),j=1,64)
      enddo
      write(21,*) '                '
      do i=1,64
      write(21,100) (ri(i,j),j=1,64)
      enddo
      close (unit=21)
      print *,'2'
      call qrc (rr,ri,tr,ti,ur,ui,n)
      open(unit=22,file=out3,access='sequential')
      write(22,*)'QR matrix '
      do i=1,64
      write(22,100) (tr(i,j),j=1,64)
100   format(2x,8f8.2)
      enddo
      write(22,*) '                '
      do i=1,64
      write(22,100) (ti(i,j),j=1,64)
      enddo
      close (unit=22)
      print*,'3'
c
c     Estimate the signal subspace dimension
c
      dmin=4096.
      min=64
      b=tr(64,64)
      s1=b
      s2=ln(b)
      do 11 i=63,1,-1
      b=tr(i,i)
      s1=s1+b
      s2=s2+ln(b)
      a=(64.-float(i))*(ln(s1/(64.-float(i))-s2
      a=a* float(n)
      a=a+float(i)*(128.-float(i))
      if (a.gt.dmin) goto 11
      dmin=a
      min=i
```

```
11      continue

        call power (ur,ui,pm,min,n)
        open(unit=23,file=out4,access='sequential')
        write(23,*)'Power method'
        do i=1,90
        jj=i-1
        write(23,*) jj,pm(i)
        enddo
        close (unit=23)
        stop
        end
c
c       This subroutine calculates the spatial spectrum
c       using MUSIC algorithm for broad band signals using
c       (BASS-ALE) estimation. See Equation (16) on Buckley
c       paper " Broad-Band Signal-Subspace Spacial-Spectrum"
c       (BASS-ALE) Estimation.""
c       Nf which control the the sample density of the location
c       vector is chosen to 5.
c
c
c     Input parametres
c         u=(ur,ui)     _matrix whose rows are the eigenvectors
c                        of the original data covariance matrix
c         n             _matrix order
c         d             _dimension of the signal subspace
c
c     Output parameter
c        pm            _Array representing the spatial spectrum
c


        subroutine power(ur,ui,pm,d,n)
        integer d
        real ur(64,64),ui(64,64),pm(360), f(5)
        real ar(64),ai(64),c(64),s(64)
        pi=acos(-1.)
        f(1)=3./40.
        f(2)=7./80
        f(3)=1./10.
        f(4)=9./80.
        f(5)=1/8.
        do 1 i=1,181
        x=(float(i)-1)*pi/180.
        theta=sin(x)/2.
        pm(i)=0.
        do 2 j=1,5
        xx=0.
        do 3 k=1,8
        d0 3 l=1,8
        a=2*pi*f(j)*((float(l)-1)*theta+(float(k)-1))
        m=(k-1)*8+l
        c(m)=cos(a)
        s(m)=sin(a)
3       continue
        do 4 m1=d+1,n
        sr=0.
        si=0.
```

```
      do 5 m2=1,n
      sr=sr+ur(m1,m2)*c(m2)-ui(m1,m2)*s(m2)
      si=si+ur(m1,m2)*s(m2)+ui(m1,m2)*c(m2)
5     continue
      xx =xx+sr**2+si**2
4     continue
      pm(i)=pm(i)+1/xx
2     continue
1     continue
      return
      end
```

REFERENCES

[1] C. H. Knapp and G. C. Carter, "The generalized correlation method for estimation of time delay, " IEEE Trans. Acoustic, Speech and Signal Processing, Vol. ASSP-24, pp. 320-327, Aug. 1976.

[2] J. Capon, "High resolution frequency-wavenumber spectrum analysis. Proc. IEEE, Vol. 57, pp. 1408-1418, Aug. 1969.

[3] J. P. Burg, "Maximum entropy spectral analysis," in Proc. 37th Ann. Int. SEG Meet. Oklahoma City, OK, 1967.

[4] R. O. Schmith, "Multiple emitter location and signal parameter estimation," IEEE Trans. on Antennas and Propagation, Vol AP-34 No. 3., pp. 276-280, Mar. 1986.

[5] R. Roy and T. Kailath, "ESPRIT-Estimation of signal parameters via rotational invariance techniques, " in proc. IEEE Trans. Acoustic, Speech and Signal Processing, Vol. 37, No. 7, pp. 984-995, July 1989.

[6] D. Spielman, A. Paulraj, "Performance analysis of the MUSIC algorithm," in proc. IEEE Conf. Acoustic, Speech and Signal Processing, Tokyo, Japan, pp 1909-1912, Apr 1986 .

[7] T. J. Shan, A. Paulraj, "On smoothed rank profile tests in eigenstructure approach to direction-of-arrival estimation," in proc. IEEE Conf. Acoustic, Speech and Signal Processing, Tokyo, Japan, pp 1905-1908, Apr 1986.

[8] G.H. Golub, C. F. Van Loan " An analysis of the total least square problem, " SIAM J. Numerical Analysis. Vol 17. N 17, December 1980.

[9] C. Y. Chen and J. A. Abraham, "Fault -tolerant systems for computations of eigenvalues and singular value" SPIE Vol.696 Advanced algorithm and architecture for signal processing, pp 228-237, 1986

[10] J. J. Dongarra and D. C. Sorensen, "On the implementation of fully parallel algorithm for symmetric eigenvalue problem", SPIE Vol. 696 Advanced algorithm and architecture for signal processing, pp 45-53, 1986

[11] J. H. Wilkinson, "The algebriac eigenvalue problem," Clarendon Press, Oxford, Chapter 4, 1965

[12] C. F. T. Tang, K. J. R. Liu and S. A. Tretter, "On systolic array for recursive complex Householder transformations with applications to array processing.", in proc. IEEE Conf. Acoustic, Speech and Signal Processing, Toronto, Canada, pp 1033-1036, Apr 1991

[13] K. J. R. Liu, S. F. Heieh, K. Yao,"Two level pipelined implementation of systolic block Householder transformations."in proc. IEEE Conf. Acoustic, Speech and Signal Processing, pp 1631-1634, Alburquerque, NM. Apr 1990

[14] S. Y. Kung VLSI array processors, Prentice Hall, Englewood Cliffs, NJ. 1987.

[15] W.Phillips and W.Robertson,"Systolic architecture for symmetric tridiagonal eigenvalue problem", IEEE International conference on systolic arrays, pp. 145-150, 1988.

[16] K.J.R.Liu and K.Yao, "Multiphase systolic architecture for spectral decomposition," 1990 International conference on parallel processing, pp I-123 to I-126.

[17] Volder J.E., "The CORDIC trigonometric computing technique", IRE transactions. electronic computers, EC-8, No.3, pp. 330-334, Joint computer conference, San Fransisco, California, 1956.

[18] Walther J.S., "A unified algorithm for elementary functions" Spring joint computer conference, AFIPS conference proceedings, 38, pp. 379-385, 1971.

[19] Gene L. Haviland and Al A. Tuszynski, "A CORDIC arithmetic processor chip", IEEE transactions on computers, Vol.c-29, No.2, pp. 68-79, February 1980.

[20] H.M. Ahmed et al, "A VLSI speech analysis chip set utilizing co-ordinate rotation arithmetic", IEEE Int. Symp. on Circuits and Systems, pp. 737-741, 1981.

[21] A. A. J. de Lange et al, "An optimal floating-point pipeline CMOS CORDIC processor", IEEE Int. Symp. on Circuits and Systems, pp. 2043-2047, 1988.

[22] D. H. Daggett, "Decimal-binary conversions in CORDIC", IRE transactions on electronic computers, EC-8, No.3, pp. 335-339, Joint computer conference, San Fransisco, California, 1956.

[23] Ahmed et al, "Highly concurrent computing structures for matrix arithmetic and signal processing", IEEE Computer, pp. 65-81, January 1982.

[24] Milos D. Ercegovac and Tomas Lang, "Implementation of fast angle calculation and rotation using on-line CORDIC", IEEE Int. Symp. on Circuits and Systems, pp. 2703-2706, 1988.

[25] S.C. Bass et al, "A bit-serial, floating-point CORDIC processor in VLSI", IEEE Int. Conf. on Acoustics, Speech and Signal Processing, V3.1, PP. 1165-1168, Toronto, Canada, 1991.

[26] Arnab K. Shaw and Ramdas Kumaresan, "Estimation of angles of arrivals of broadband signals", IEEE ICASSP-87, pp.2296-2299, 1987.

[27] Guaning Su and Martin Morf, "modal decomposition signal subspace algorithms", IEEE Trans. on Acoustics, Speech, and Signal Processing, VOL. ASSP-34, No. 3, pp. 585-602, June 1986.

[28] Bjorn Ottersten and Thomas Kailath, "Direction-of-arrival estimation for wide-band signals using the ESPRIT algorithm", IEEE Trans. on Acoustics, Speech, and Signal Processing, VOL. 38, No. 2, pp. 317-327, February 1990.

[29] Kevin M. Buckley and LLoyd J. Griffiths, "Broad-band signal-subspace spatial-spectrum (BASE-ALE) estimation", IEEE Trans. on Acoustics, Speech, and Signal Processing, VOL. 36, No. 7, July 1988.

[30] G. C. Carter, [Editor], "Special issue on time-delay estimation", IEEE Trans. on ASSP, VOL. 29, No. 3, 1981.

[31] C. H. Knapp and G. C. Carter, "The generalized correlation...", IEEE Trans. ASSP, VOL. 24, No. 4, pp. 320-237, 1976.

[32] W. J. Bangs and P. Schultheiss, "Space-Time processing...", in *Signal Processing*, J. W. R. Griffiths et al, Eds. New York, Academic Press, pp. 577-590, 1973.

[33] W. R. Hahn and S. A. Tretter, "Optimum processing for ...", IEEE Trans. IT, VOL. 19, No. 5, pp. 608-614.

[34] M. Wax and T. Kailath, "Optimum localizations of multiple source by passive arrays", in proc. IEEE Trans. Acoustic, Speech and Signal Processing vol. ASSP-31, No. 5, pp. 1210-1218, Oct. 1983.

[35] M. Morf. et al, "Investigation of new algorithms ...", DARPA Tech. Rept., No. M-355-1.

[36] B. Porat and B. Frienlander, "Estimation of spatial and spectral parameters of multiple sources", IEEE Trans. on info. theory, vol. IT-29, pp. 412-425, May 1983.

[37] A. Nehorai, G. Su, M. Morf, "Estimation of time difference of arrivals for multiple ARMA sources by pole decomposition", in proc. IEEE Trans. Acoustic, Speech and Signal Processing, vol. ASSP-31, pp. 1478-1491, Dec. 1983.

[38] M. Wax T. J. Shan and T.Kailath, "Spatio-temporal spectral analysis by eigenstructure method", in proc. IEEE Trans. Acoustic, Speech and Signal Processing, vol. ASSP-32, No. 4, Aug. 1984.

[39] H. Wang and M. Kaveh, "Estimation of angles-of -arrival for wide-band sources", in proc. IEEE Trans. Acoustic, Speech and Signal Processing, pp. 7.5.1-7.5.4, Mar. 19-21, 1984.

[40] H. Wang and M. Kaveh, "Coherent Signal Subspace processing for the detection and estimation of angle of arrival of multiple wide-band sources", in proc. IEEE Trans. Acoustic, Speech and Signal Processing, vol. ASSP-33, No. 4, pp. 823-831, Aug. 1985.

[41] T. L. Henderson, "Rank Reduction for Broadband ...", in proc. *19th Asilomar Conf. on Circ., Syst. & Comp.*, Nov. 1985.